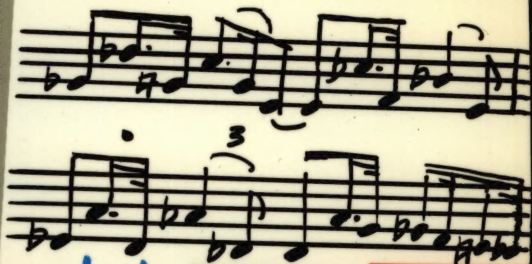
Logisch Logo

Auke Sikma



ACADEMIC SERVICE

LOGISCH LOGO

Logisch Logo

Auke Sikma

ACADEMIC SERVICE

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Sikma, Auke

Logisch Logo / Auke Sikma ; [ill. André de Regt]. - Den Haag : Academic Service. - Ill.
ISBN 90-6233-203-X
ISSO 365.3 SVS 8.12.3 UDC 800.92 Logo UGI 650
Trefw.: Logo (programmeertaal).

Uitgegeven door: Academic Service
Postbus 81
2870 AB Schoonhoven
Zetwerk: multitASk
Illustraties: André de Regt
Omslagontwerp: André de Regt
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam
ISBN 90 6233 203 X

VOORWOORD

Er zijn mensen die menen dat je niet over Logo moet schrijven, laat staan dat je erover moet lezen. 'Logo moet je ervaren' is het motto. Met dat laatste ben ik het van harte eens. Wellicht kan dit boek u helpen om Logo te ervaren.

Logo, de programmeertaal, is tegelijk een visie op onderwijs. Ik denk dat het programmeren in Logo voor veel mensen een openbaring kan zijn.

Zoals u weet zijn computers in feite slechts 'domme' apparaten. Uit zichzelf zal de computer niets ondernemen. Om de computer te vertellen wat hij doen moet hebben we een taal nodig, een taal om met de computer te communiceren. En eigenlijk verstaat de computer maar één taal: de taal van zijn 'hart', de CVE (Centrale Verwerkings Eenheid).

Deze 'moedertaal' van de computer bestaat slechts uit énen en nullen. Het zou bijzonder onplezierig zijn wanneer we slechts 'zo' met de computer konden communiceren. We zouden het spoor snel bijster raken. Om iets aan dit probleem te doen zijn er vele 'hogere' programmeertalen ontwikkeld. Door de computer echter wordt iedere opdracht vanuit een 'hogere' taal ogenblikkelijk weer veranderd in een serie énen en nullen.

Er zijn zeer vele 'hogere' programmeertalen: Ada, BASIC, Pascal, PROLOG, LISP, COBOL, ALGOL, APL, Pilot, C, Smalltalk, Forth, FORTRAN, Action, en dus ook Logo.

Dit lijstje van computertalen is niet compleet, en bovendien bestaan er van iedere taal nog vele dialecten. Microsoft BASIC is anders dan Extended BASIC en ook BBC BASIC is anders dan Commodore BASIC. Ziet u door de bomen het bos niet meer? Ach, het valt met Logo allemaal wel mee, die verschillen. Natuurlijk zijn ze er wel, maar ze zijn aanmerkelijk kleiner dan bij BASIC bijvoorbeeld. Zelf heb ik met verschillende Logo-versies gewerkt.

De procedures in dit boek zijn geschreven in LCSI Logo (voor Atari), maar ze kunnen geschikt worden gemaakt voor iedere Logo. Daarvoor vindt u achterin dit boek een vergelijkingstabel waarin de meestgebruikte Logo-woorden van de meestgebruikte Logo-versies staan. Het gaat om de Logo-versies voor de volgende merken computers: Atari 8-bits en 16-bits, Apple, BBC/Acorn, Commodore, MSX en Spectrum. U zult zien dat de diverse Logo's weinig van elkaar verschillen, al hebben ze elk hun eigen mogelijkheden.

En dan over dit boek. Ik heb geenszins de pretentie volledig te zijn. Dat kan ook niet, omdat ik van mening ben dat Logo op zijn 'waardevolst' is in de hand van zijn gebruiker(ster). Ik kom hier later op terug. De bedoeling van dit boek is een inleiding, een wegwijzer, te zijn in het gebruik van de taal Logo. *Logisch Logo* wil u bouwstenen aanreiken waarmee u uw eigen Logo-wereld bouwt.

Ik raad u aan bij het lezen geen hoofdstukken over te slaan. Vaak worden er begrippen, opdrachten en/of procedures gebruikt die eerder zijn uitgelegd.

Het is belangrijk (en leuk!) om via de programma's en procedures in dit boek greep te krijgen op Logo, om zo, in eigen omgeving, op geheel eigen wijze inspiratie te putten uit en bevrediging te vinden in het werken met Logo.

Dat Logo een uitstekend middel is om analytisch en probleemoplossend te leren denken moet toch reden genoeg zijn om er zo snel mogelijk mee te willen beginnen. Moge dit boek daaraan een bescheiden bijdrage leveren.

VERANTWOORDING

Het was de bedoeling een zo oorspronkelijk mogelijk boek over de programmeertaal Logo te schrijven. Er is echter niet aan te ontkomen om voor ideeën, programmeervaardigheden en hulpprocedures te putten uit andere, veelal Amerikaanse, Logo-bronnen. Het een en ander is echter steeds naar eigen inzicht bewerkt, veranderd en aangepast. Bepaalde gedeelten uit het boek, waaronder enkele procedures, zijn door mij reeds eerder in verschillende bladen gepubliceerd.

De illustraties, alle origineel en speciaal voor deze uitgave gemaakt, zijn van André de Regt. Als basis voor de illustraties is steeds gekozen voor de Turtle. De Turtle is immers het herkenbare, zeg maar het handelsmerk, van Logo. Als zodanig verschijnt de Turtle ook in de Logo Taal- en Muziekwereld, twee Logo-werelden die niets met de Turtle van doen hebben.

De 'Turtle Graphics' illustraties zijn met toestemming overgenomen uit de werkboekjes van het Logo Centrum Ede.

Een enkele opmerking nog over de afgedrukte tekst van Logo-procedures:

- druktechnisch bleek het niet mogelijk de lange procedureregels achter elkaar op één regel af te drukken. Dergelijke 'lange' regels zijn daarom gesplitst. Bijvoorbeeld:

```
IF :INPUT = FIRST FIRST :DRAAIOM [OP LAST FIRST :D
RAAIOM] [OP VERGELIJK :INPUT BF :DRAAIOM]
```

Hoewel deze procedureregel op twee regels is afgedrukt, is het toch één procedureregel. Pas na het intikken van deze gehele regel mag de RETURN-toets ingedrukt worden.

- Onthoud:
  Iedere procedureregel begint met een opdracht (of met het aanroepen van een subprocedure).
  Iedere procedureregel wordt afgesloten door een druk op de RETURN-toets.

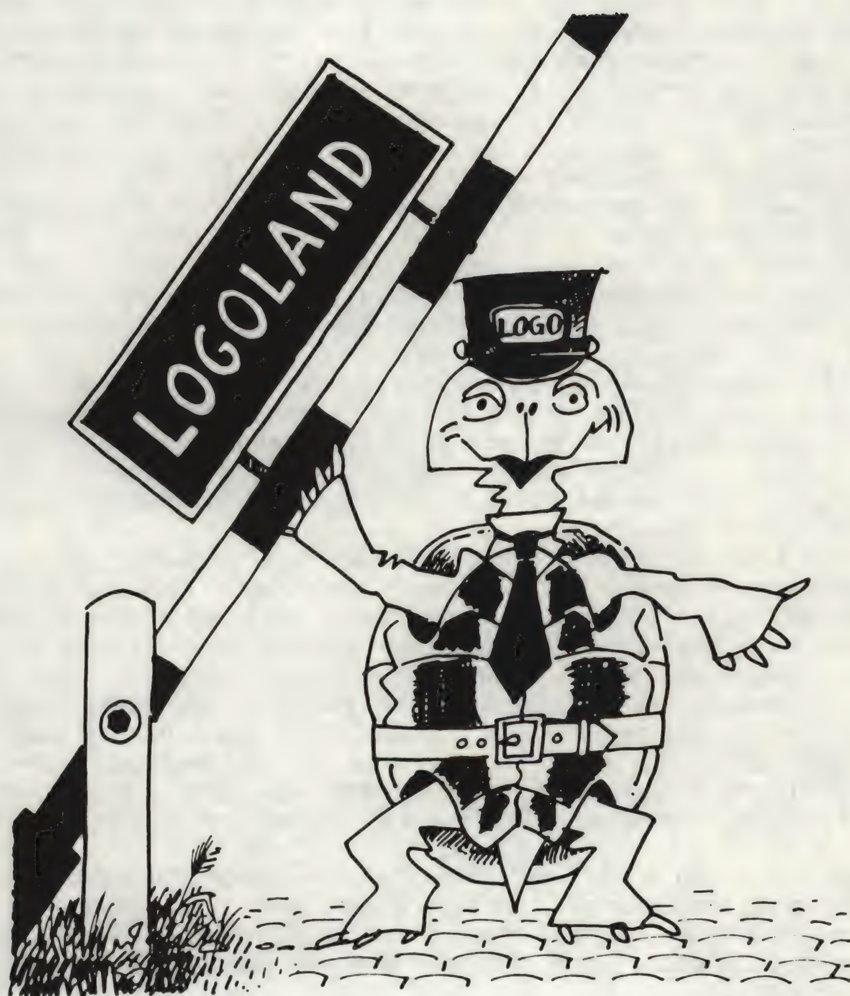
Als u dit in acht neemt zullen alle voorbeeldprocedures (eventueel aangepast aan uw eigen Logo-versie) moeten draaien.

Rinsumageest
juni 1986

Auke Sikma

INHOUD

*Hoofdstuk 1 – **WELKOM BIJ LOGO***



1.1 Wat is Logo..., een inleiding

De programmeertaal Logo werd aan het eind van de zestiger jaren ontwikkeld op het 'Massachusetts Institute of Technology' in Amerika. Een van de mensen die zeer nauw bij het onderzoek betrokken waren was Seymour Papert. Deze Seymour Papert wordt door velen dan ook gezien als de geestelijke vader van Logo. Een journalist heeft Papert eens de *Spiritual father of the Logo-Movement* genoemd.

U kunt echter gerust zijn: Logo is geen geloof en ook geen sekte. Geen geestelijke stroming, of een manier van denken. Door met Logo te werken leert u echter wel op een andere manier denken. Met Logo als 'instrument', als 'gereedschap' leert u analytisch en 'probleemoplossend' denken. Later wordt hier uitgebreid op ingegaan.

Seymour Papert omschrijft Logo aldus: *"A versatile computer language that provides a friendly introduction to programming, a serious tool for advanced programmers and a medium for educational discovery."* Dit citaat komt uit een interview met Seymour Papert dat heeft gestaan in het blad *Atari Connection*, herfst 1983.

Paperts boek *Mindstorms, children, computers and powerful ideas* beschrijft de filosofie achter Logo. In het Nederlands is dit boek verkrijgbaar onder de titel *Computers en kinderen* (uitgave Bert Bakker, Amsterdam). In *Mindstorms* beschrijft Papert de ontwikkeling van Logo, en welke ideeën aan de taal ten grondslag liggen.

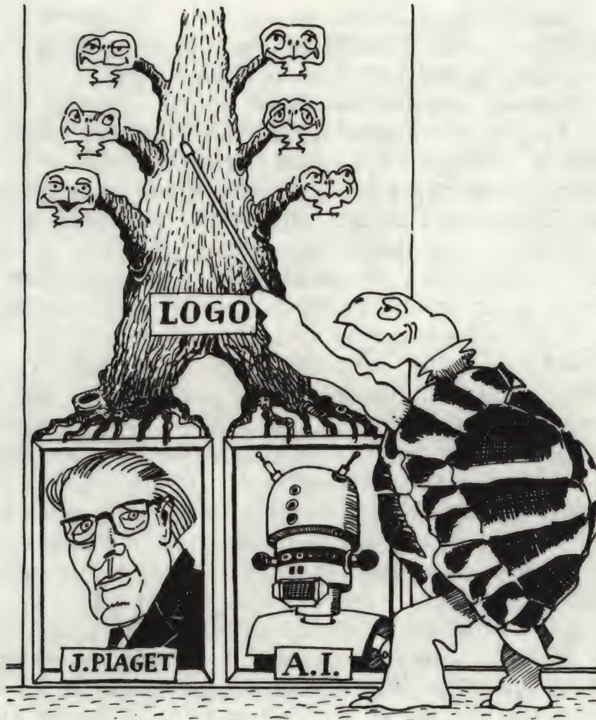
Aanvankelijk is Logo ontwikkeld met het oog op kinderen. Logo moest een taal worden zonder drempel, zeer gebruikersvriendelijk. In de eerste Logo-versies werd de nadruk gelegd op het gebruik van woorden en lijsten. Het is dan ook niet zo verwonderlijk dat Logo een nauwe verwantschap heeft met een professionele taal als LISP (LISt Processing).

Om Logo echter voor een groot publiek gebruikersvriendelijk en toegankelijk te maken werd de Turtle (Schildpad) geïntroduceerd. De Turtle is een bewegende en tekenende 'vloerrobot' of een figuur op het beeldscherm. In sommige Logo-versies echter is deze Turtle reeds verworden tot een simpel driehoekje.

Het maken van tekeningen met de Turtle wordt Turtle Graphics genoemd. Hoewel aanvankelijk bedacht als opstapje tot het 'Logo-programmeren' zijn met de Turtle Graphics toch zeer geavanceerde afbeeldingen te maken. Papert zelf noemt zijn Turtle Graphics een aanzet tot 'proces-georiënteerde en dynamische wiskunde'.

Logo komt voort uit het onderzoek in kunstmatige intelligentie enerzijds en de leertheorieën van Jean Piaget anderzijds. Er is over beide zaken reeds veel geschreven en het is niet nodig er in dit boek diep op in te gaan. Toch lijkt het zinvol dit onderwerp kort te bespreken.

1.2 De wortels van Logo: kunstmatige intelligentie (AI) en Jean Piaget



Een 'zelf-denkende' computer, die concludeert, constateert en reageert, die gewoon met de stem, dus zonder specifieke 'computertalen', aangesproken kan worden, en die ons adviseert en helpt bij het nemen van beslissingen. Is dit fantasie? Science fiction wellicht? Nee, het is het dagelijkse streven van de vele onderzoekers op het gebied van kunstmatige intelligentie (artificial intelligence).

Kunstmatige intelligentie is een tak van computerwetenschap die zich bezighoudt met de 'begrippen en methodieken voor het maken van gevolgtrekkingen door een computer'. Deze wetenschap spitst zich dus toe op de zogenaamde 'denkende' machines. Een en ander wordt aangeduid met de term: vijfde-generatie computers of KIPS (Knowledge Information Processing Systems).

Wellicht is de term 'vijfde-generatie' u onbekend. Hiermee wordt bedoeld dat deze generatie computers niet slechts in staat is tot verwerking van gegevens, maar tevens tot beredeneerde manipulatie van gegevens. De vier eerdere generaties computers zijn:

a. computers op grond van elektrische vacuümbuizen;
b. getransistoriseerde computers;
c. computers met geïntegreerde schakelingen (IC's);
d. computers met VLSI (Very Large Scale Integration).

Vier generaties lang was het enige wat computers bezaten: informatie. Maar 'informatie bezitten' is niet hetzelfde als intelligentie. Dat zien we overigens niet alleen bij machines.

Intelligentie komt van het Latijnse woord 'legere' (wat o.a. kiezen betekent). En, zo reageerden de onderzoekers, om intelligentie te creëren hebben we niet voldoende aan kennis en informatie alleen. Iedere computer kan informatie opnemen, maar is daarmee nog niet intelligent. Intelligentie is onder meer het manipuleren van informatie. Ter illustratie het volgende voorbeeld.

Stel u wilt, enthousiast gemaakt door dit boek, nog meer weten over Logo en u besluit om hierover nog een boek aan te schaffen. Op grond van kennis en informatie zult u dan niet naar een platenwinkel of doe-het-zelf-zaak gaan. Op grond van ervaring (empirisch denken) en logisch denken (heuristiek) zult u naar een boekwinkel gaan, waarvan u aanneemt dat ze het boek in voorraad hebben; en u zult wellicht meteen bij de informaticaboeken kijken en zo nodig vragen het boek voor u te bestellen.

Op grond van informatie en kennis had u het boek niet of nauwelijks kunnen kopen. Op grond van de door u beheerste kennis, alsmede het vermogen deze kennis door middel van empirisch denken en heuristiek te manipuleren kunt u intelligent genoemd worden.

Maar ... hoe zit het dan met machines? Is het mogelijk denkende en intelligente machines te ontwerpen? Veel mensen schrikt deze vraag af, en tegelijkertijd is dat niet zo verwonderlijk.

Op de voorplaat van de *Haagse Post* van 22 maart 1986 staat een illustratie die treffend de angst van mensen voor 'denkende machines' laat zien. De tekening, van de Amerikaan Keith Haring, laat ons een woedende massa zien rond een 'denkende' computer. Dat de computer denkt zien we door de op het beeldscherm afgebeelde hersenen. De computer wordt boven de massa uitgetild door een, in contrasterende kleuren getekende, figuur. Beeldt deze figuur wellicht de elite uit die zich met deze wetenschap bezighoudt? Wie weet ... Maar het is wél duidelijk dat deze tekening de gedachten van velen over kunstmatige intelligentie op voortreffelijke wijze illustreert.

Waarom die angst? We hebben, in meerdere of mindere mate, reeds geaccepteerd dat machines ons fysiek overtreffen. Of het nu gaat om kanalen graven, bomen omzagen, betonpalen verplaatsen of cement maken, steeds weer blijkt dat machines in staat zijn deze werkzaamheden sneller, nauwkeuriger en dus 'beter' te doen dan wij zelf ooit zouden kunnen. Maar of wij nu machines kunnen maken die ons ook verstandelijk overtreffen ...? Op verstandelijk gebied zijn wij de machines namelijk nog steeds de baas.

Een computer is volledig afhankelijk van de gegevens die wij hem verstrekken. Hoewel het onderzoek naar kunstmatige intelligentie

nog verschillende problemen te overwinnen heeft, wordt er nu reeds geëxperimenteerd met 'denkende machines'.

Wellicht zegt de term 'expertsysteem' u iets. Een expertsysteem is een computerprogramma dat bijvoorbeeld gebruikt kan worden bij het stellen van een medische diagnose. De computer zal dan op basis van het expertsysteem naast het combineren van theoretische kennis ook 'empirisch' gericht zijn. Dat wil zeggen dat de computer als het ware 'leert' van de ervaring. Indien bij bepaalde symptomen in 60% van de keren diagnose X wordt gesteld, zal de computer dit gegeven gebruiken bij het stellen van een volgende diagnose op grond van dezelfde symptomen. Aldus benadert de computer enigszins de deskundigheid van de expert, de arts of specialist.

De computer mist echter nog steeds de logica (heuristisch denken) om tot een bevredigende oplossing te komen. Zo zou de computer op grond van de beschikbare informatie een volstrekt onlogisch antwoord kunnen geven.

Welnu, de computertalen in de kunstmatige intelligentie zijn Prolog en LISP. Logo is een directe afleiding van LISP.

Seymour Papert is, hoewel het meeste onderzoek in Japan gebeurt, lange tijd betrokken geweest bij het onderzoek aan het Artificial Intelligence Laboratory in Massachusetts. Vanuit dit onderzoek en geïnspireerd door het denken over de psychologie van Jean Piaget ontstond aan het eind van de zestiger jaren de programmeertaal Logo.

Laten we maar eens kijken welke ideeën van Jean Piaget Seymour Papert inspireerden. Enkele uitspraken van Papert die zijn visie op het onderwijs weergeven:

- Het huidige schoolsysteem geeft de leerkrachten weinig gelegenheid om de kinderen werkelijk iets te leren. In plaats daarvan moeten zij de kinderen volstoppen met achterhaalde kennis, onnuttige feiten en dogmatische ideologieën.

- Scholen zullen een leeromgeving moeten ontwikkelen waarbinnen kinderen geen kennis opgedrongen wordt, maar waar hun, al doende, kennis en informatie aangeboden wordt. Voor het ontwikkelen van zo'n leeromgeving zijn, in een technologische maatschappij als de onze, computers onontbeerlijk.

- Passen de scholen zich niet aan deze veranderde kijk op het leren aan, dan is hun lot bezegeld: ze zullen net als dinosaurussen verdwijnen.

- De ontwikkeling van kinderen gaat nauw samen met het omgaan met nieuwe situaties. Daarom zal het kind pas werkelijk iets leren wanneer het zelf met nieuwe situaties omgaat en ze niet slechts observeert. Om dergelijke nieuwe situaties te creëren zullen we binnen het onderwijs zogenaamde 'leeromgevingen' moeten scheppen.

- Binnen een 'leeromgeving' beïnvloedt het kind zijn omgeving en het wordt er zelf door beïnvloed.
  Binnen een 'leeromgeving' wordt een kind niet gestraft voor origineel en non-conformistisch denken maar wordt dit juist gestimuleerd.
  Binnen een 'leeromgeving' is het het kind dat manipuleert en niet de omgeving.

Seymour Papert geeft toe sterk te zijn beïnvloed door de ideeën van de Zwitserse psycholoog Jean Piaget. Regelmatig spreekt Papert over het zogenaamde 'Piagetiaanse leren'. Onder het 'Piagetiaanse leren' wordt dan verstaan: het natuurlijke en spontane leren van mensen in wisselwerking met hun omgeving.

Vaak wordt dit 'leren' geplaatst tegenover het traditionele leren volgens een bepaald leerplan (wat kenmerkend is voor het huidige onderwijssysteem). Het is dan ook niet verwonderlijk dat Papert zich herhaaldelijk afzet tegen het traditionele schoolsysteem.

In nauwe relatie met de leertheorieën van Piaget creëerde Papert binnen Logo de zogenaamde leeromgevingen. Papert zelf noemt deze leeromgevingen: 'Microwerelden: Broedmachines voor Kennis'. Zonder van dit boek een psychologisch werk te willen maken nog een enkele toelichting op het begrip 'leeromgeving'.

Het opdoen van kennis binnen een 'leeromgeving' wordt door Papert nauw verbonden met het proces dat in Piagets psychologie 'assimilatie' wordt genoemd. Ter verduidelijking moge het volgende dienen. Als iemand iets nieuws tegenkomt, een nieuwe ervaring opdoet, zal dit met de reeds aanwezige kennis in verband worden gebracht na te zijn waargenomen. De aldus verkregen nieuwe kennis zal vervolgens gehanteerd worden om met volgende situaties om te gaan. Met andere woorden: het nieuwe verwerken in het oude om vervolgens met het nieuwste te gaan werken.

Dit proces dat, omdat het nieuwe vaak tegenstrijdig is met het oude, soms conflictsituaties oplevert wordt assimilatie genoemd. En deze assimilatie, dit proces van opdoen van nieuwe kennis om vervolgens zelf hiermee te manipuleren, is de diepere gedachte achter Paperts 'leeromgeving'.

1.3 Wat maakt Logo zo speciaal?

Om dit te illustreren beginnen we meteen maar met het bekijken van twee korte Logo-programmaatjes. Later worden deze en andere programma's in aparte hoofdstukken nader besproken.

```
EDIT "FIGUUR :ZIJDE :HOEK :VERGROTING
FORWARD :ZIJDE
RIGHT :HOEK
FIGUUR :ZIJDE + :VERGROTING  :HOEK :VERGROTING
END
```

Het resultaat van dit programma (met de drie variabelen :ZIJDE, :HOEK en :VERGROTING) is een figuur die steeds groter en groter wordt. Aan het eind van de procedure treedt een herhaling (recursie) in werking omdat de procedure als het ware zichzelf opnieuw aanroept (later hierover meer).

```
EDIT "PRAATJE
PRINT [HOE IS HET WEER VANDAAG ?]
MAKE "ANTWOORD READLIST
IF :ANTWOORD = [MOOI] [PRINT [DAN GAAN
WE BUITEN ZITTEN]][PRINT [O , JA ...]
WAIT 50
PRINT [NOU IK GA MAAR WEER EENS]
PRINT [TOT ZIENS]
END
```

Dit programma kan resulteren in het volgende 'praatje':

```
HOE IS HET WEER VANDAAG ?
?MOOI
DAN GAAN WE BUITEN ZITTEN
NOU IK GA MAAR WEER EENS
TOT ZIENS
```

of

```
HOE IS HET WEER VANDAAG ?
?AFSCHUWELIJK , HET REGENT NOG STEEDS
O , JA ...
NOU IK GA MAAR WEER EENS
TOT ZIENS
```

Aan deze programma's is goed te zien dat Logo niet zomaar een taal is. Logo heeft enkele zeer belangrijke kenmerken die nu achtereenvolgens besproken worden. Hier zijn de vier belangrijkste kenmerken:

a. Logo is interactief;
b. Logo is procedureel;
c. Logo is uit te breiden met door u zelf gedefinieerde opdrachten en procedures;
d. Logo beschikt over de mogelijkheid dat procedures zichzelf aanroepen (recursie).

Er volgt nu een beschrijving van deze vier Logo-aspecten onder andere aan de hand van de twee bovenstaande voorbeelden.

a. Logo is interactief

In tegenstelling tot veel andere talen is Logo interactief. Dat wil zeggen dat het mogelijk is om met Logo iets te proberen en daarvan meteen het resultaat te zien.

Als we bijvoorbeeld FORWARD 100 typen en daarna op de return- (of enter-) toets drukken, zien we ogenblikkelijk het resultaat van deze opdracht: de Turtle loopt 100 stapjes vooruit en tekent zo een lijn op het beeldscherm.

Als we typen PRINT "HALLO en daarna return, zien we ogenblikkelijk de boodschap HALLO op het scherm.

Ook een iets ingewikkelder opdracht wordt direct uitgevoerd. De tafel van 127? Dat is geen enkel probleem, kijk maar eens naar deze eenregelige opdracht:

```
MAKE "A 127 REPEAT 10 [PR :A MAKE "A :A + 127]
```

Natuurlijk krijgen we zo alleen de uitkomsten en niet de bewerkingen zelf in beeld, maar daarover later meer.

Omdat Logo interactief is, is het een uitermate geschikte taal voor kinderen. De kinderen zien immers ogenblikkelijk het resultaat van hun opdracht.

b. Logo is procedureel

We noemen Logo een proceduretaal. Dit betekent dat een 'groot' programmeerprobleem in Logo als het ware wordt opgesplitst in vele 'kleintjes'.

Om maar eens iets te noemen: er moet een huis getekend worden. Laten we zeggen dat een huis onder andere bestaat uit muren, ramen, een deur, een dak en een schoorsteen. Schrijven we nu een BASIC-programma voor het tekenen van zo'n huis, dan wordt het een heel verhaal. Weliswaar kunnen we de verschillende onderdelen van het huis via subroutines laten tekenen, maar toch blijft het min of meer ongestructureerd.

Schrijven we ditzelfde programma in Logo, dan gaan we als volgt te werk. Eerst maken we een programmaatje voor de muren. Werkt het? Mooi, dan gaan we verder met een programmaatje voor de ramen, enzovoorts. Uiteindelijk krijgen we dan één hoofdprocedure die we HUIS zullen noemen en vijf subroutines, te weten: MUREN, RAMEN, DEUR, DAK en SCHOORSTEEN. Het Logo-programma kan er zo uitzien:

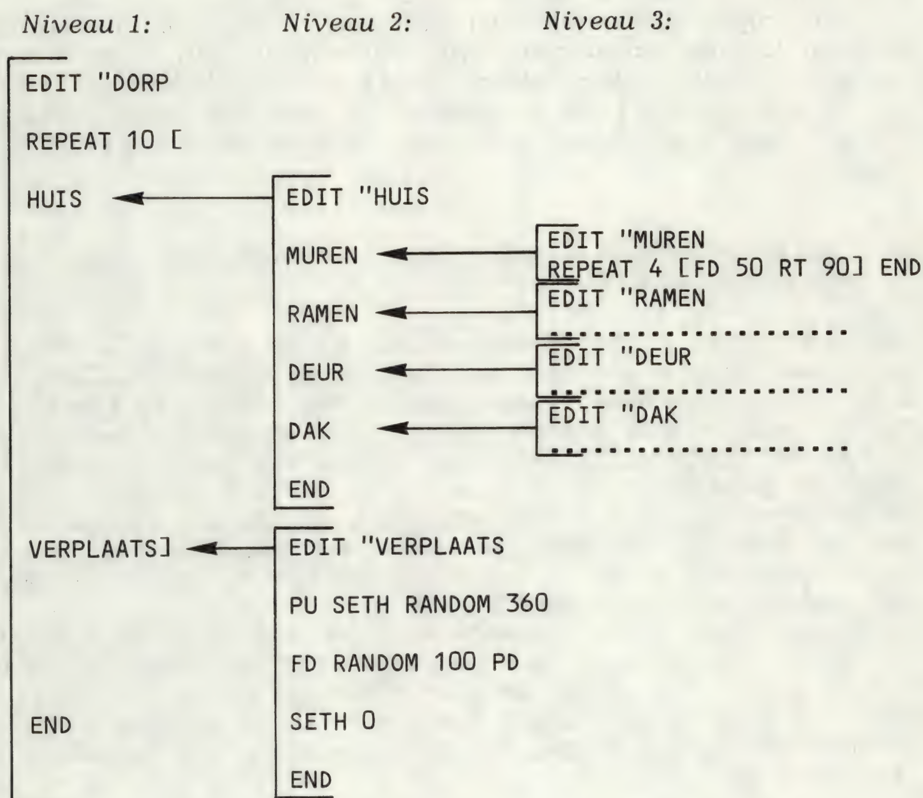
```
EDIT "HUIS
MUREN
RAMEN
DEUR
DAK
SCHOORSTEEN
END
```

Voordat dit programma foutloos werkt zullen natuurlijk wel eerst de subprocedures MUREN, RAMEN, DEUR, DAK en SCHOORSTEEN geschreven moeten worden. Een voordeel is dat we de procedures overal kunnen gebruiken. Willen we bijvoorbeeld een kasteel maken, dan zouden we daarbij de procedure RAMEN rustig kunnen gebruiken.

Iedere procedure heeft zijn specifieke taak. De procedure HUIS

heeft tot taak een huis op het scherm te laten verschijnen.

We hebben HUIS nu gebruikt als hoofdprocedure; HUIS kunnen we echter ook als subprocedure, dus als onderdeel van een andere hoofdprocedure, gebruiken. We kunnen bijvoorbeeld aan een hoofdprocedure "DORP denken, een tekening met meerdere huizen. Wellicht kan het volgende overzicht verhelderend werken:



Op niveau 3 vinden we de procedure MUREN. Deze wordt op niveau 2 gebruikt in de procedure HUIS. De procedure HUIS wordt op zijn beurt op niveau 1 weer gebruikt in de procedure DORP. Eigenlijk is het heel eenvoudig.

c. Logo is uitbreidbaar

De taal Logo heeft de beschikking over veel opdrachten en bewerkingen. Logo-woorden noemen we 'primitieven' (primitives). Het mooie van Logo is nu, dat iedereen zijn of haar eigen primitieven kan schrijven. Eigenlijk zagen we dat al in het schema hierboven. Hoewel Logo niet beschikt over een woord als "VERPLAATS, hebben

we dit zelf aan de standaardopdrachten, zoals FORWARD en PRINT, toegevoegd. Wanneer we VERPLAATS intypen, zetten we de procedure VERPLAATS in werking. Logo accepteert de naam van een zelfgedefinieerde procedure, zoals VERPLAATS, op dezelfde manier als de al in Logo aanwezige opdrachten, zoals bijvoorbeeld PRINT.

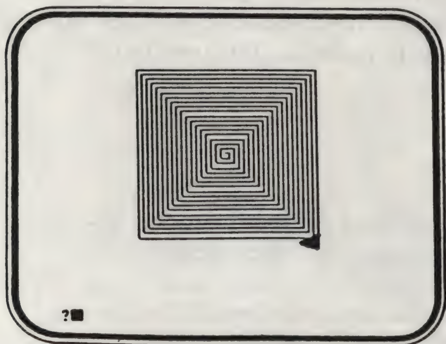
Op deze manier kan iedere gebruiker zijn eigen taal maken en die woorden en hulpprogramma's aan Logo toevoegen die voor hem of haar handig en gemakkelijk in het gebruik zijn.

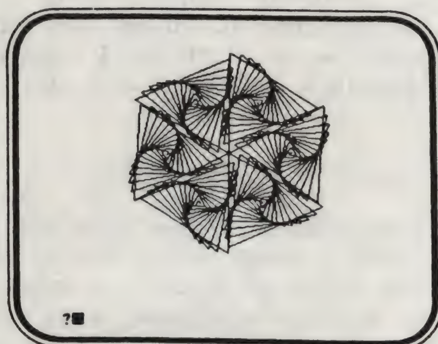
Verderop in dit boek worden zeer veel nuttige hulpprocedures gegeven die iedereen naar hartelust kan veranderen, aanpassen en toevoegen aan het Logo-systeem. Misschien is de NederLogo (zie verderop in dit boek) wel de duidelijkste illustratie van deze mogelijkheid. Deze mogelijkheid is overigens ook in een taal als Forth aanwezig.

d. Logo beschikt over de mogelijkheid dat procedures zichzelf aanroepen (recursie)

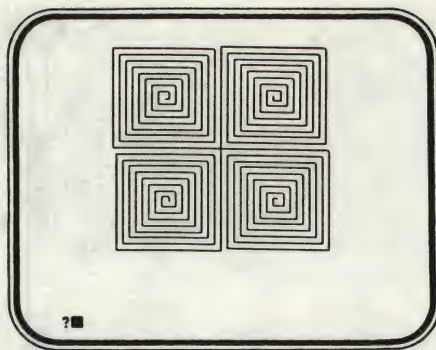
Logo-procedures kunnen recursief zijn. Recursie is een zeer krachtig aspect van Logo met zeer veel gebruiksmogelijkheden. Maar wat is nu precies 'recursie'? Hoewel later, bij de bespreking van de Turtle Graphics, nog expliciet op recursie ingegaan zal worden, volgt nu vast een korte omschrijving van het begrip. Als de computer (in Logo) een bepaalde procedure aan het uitvoeren is en hij komt daarin zijn eigen naam (die van de procedure) tegen, dan gaat de computer de procedure nogmaals uitvoeren. De procedure roept zichzelf als het ware aan. In BASIC zou je het een soort LUS kunnen noemen. Op zich natuurlijk niets spectaculairs.

De procedure roept zichzelf steeds opnieuw aan en dit kan, als we niet ingrijpen, eeuwig zo doorgaan. Het bijzondere schuilt in het feit dat de voorbeeldprocedure steeds een iets andere uitwerking heeft. Op deze manier kunnen we tot prachtige resultaten komen. De onderstaande tekeningen zijn bijvoorbeeld allemaal gemaakt met recursieve procedures.









Later, bij de beschrijving van de afzonderlijke Logo-werelden, zullen we het begrip recursie nog vele malen tegenkomen.

1.4 De vier Logo-werelden

De mogelijkheden van Logo als computertaal zullen in dit boek besproken worden aan de hand van een indeling in leeromgevingen. De term 'leeromgeving' is, zoals we al eerder zagen, afkomstig uit de ontwikkelingspsychologie van Jean Piaget.

De term leeromgeving laat een stukje zien van de visie die de makers van Logo op het onderwijs hadden. Ter illustratie het volgende citaat van Seymour Papert: "Ik geloof dat de alom aanwezige computer ons in staat zal stellen de leeromgeving buiten de schoolklas zo te veranderen dat veel, zo niet alle kennis die de school tegenwoordig met veel pijn en moeite en met zo weinig succes probeert bij te brengen, geleerd kan worden zoals het kind leert spreken, dus moeiteloos, met succes en zonder georganiseerde instructie."

Op Paperts onderwijsvisie zullen we later nog ingaan. We beperken ons eerst tot de leeromgevingen, ook wel Logo-werelden genoemd. De vier Logo-werelden laten elk een ander aspect van Logo zien. Ze kunnen afzonderlijk, in samenhang of tegelijkertijd worden aangesproken. Het zijn:

a. De Schildpadwereld
b. De Taalwereld
c. De Muziekwereld
d. De Sprokenwereld



a. De Schildpadwereld

Dit is de wereld van de Turtle Graphics en dit is wat de meeste mensen van Logo kennen. Het is de wereld van het tekenen en rekenen, van lijnen, hoeken, kleuren en vormen. De schildpadwereld is een zeer uitgebreide wereld en biedt veel mogelijkheden.

Oorspronkelijk is de taal Logo ontwikkeld zonder Turtle Graphics. In de schildpadwereld wordt gesproken over de 'Turtle Geometry', de Turtle-wiskunde: tekenen en rekenen met de Turtle als uitgangspunt. Dit is, en zeker niet alleen voor kinderen, een zeer uitnodigende wereld.

b. De Taalwereld

In de Logo-taalwereld zien we pas echt de zeer nauwe verwantschap met de kunstmatige intelligentie: manipuleren met woorden, lijsten en variabelen, of het nu om een 'kaartenbak', een 'adventure', een 'vertaalprogramma' of een 'tekstverwerker' gaat. Deze voorbeelden komen in de desbetreffende hoofdstukken aan de orde.

Logo-taal is de wereld van LISP. LISP (de taal waarin de meeste programma's met kunstmatige intelligentie zijn geschreven) is een taal die in eerste instantie is ontworpen om lijsten met symbolen te manipuleren. In de Logo-taalwereld herkennen we deze toepassingen.

c. *De Muziekwereld*

Op zichzelf beschouwd is deze Logo-wereld niet zo bijzonder. Wat betreft de muzikale mogelijkheden hebben andere programmeertalen meer te bieden. Denk maar eens aan Simon's BASIC (van Commodore) of een assembleertaal. Moeten we deze wereld dan maar overslaan?

Het antwoord zou 'ja' geweest zijn, ware het niet dat we ook hier de unieke mogelijkheden van Logo kunnen hanteren. Binnen Logo, dus ook binnen de muziekwereld, heeft elke gebruiker de mogelijkheid zijn eigen wereld, zijn microwereld, te ontwerpen. Omdat Logo uitbreidbaar is en uit procedures is opgebouwd kan iedere gebruiker zijn eigen accenten leggen, daar waar zijn interesse ligt. Toegegeven, Logo-muziek mag dan niet de uitgebreidste muziekprogrammeermogelijkheid zijn, het is wel bijzonder gebruikersvriendelijk en stelt iedereen in staat op zijn eigen niveau en naar eigen interesse te werken.

Illustraties worden gegeven in het hoofdstuk Logo-muziek.

d. *De Sprokenwereld*

De sprokenwereld is de wereld van de animaties, van de bewegende beelden. Ten aanzien van deze Logo-wereld geldt eigenlijk precies hetzelfde verhaal als voor de Logo-muziekwereld. Ook in BASIC en assembleertaal zijn veel animatiemogelijkheden, maar Logo is gebruikersvriendelijker. De animatiemogelijkheden in Logo zijn zeer uitgebreid; we noemen het ontwerpen van sproken (in BASIC 'sprites' genoemd), de 'When Demons' en de 'Collision'.

Misschien zijn dit voor u nog wat vage begrippen, maar later in het boek zullen ze zeer uitvoerig besproken worden.

Dit waren de vier Logo-werelden, de leeromgevingen. Het is zeer goed mogelijk in Logo programma's te schrijven waarin gebruik gemaakt wordt van alle vier de werelden. Denk bijvoorbeeld maar eens aan een 'adventure'. Dan kunnen we:

- Turtle Graphics gebruiken voor de illustraties van het verhaal;
- Taal gebruiken voor het vraag- en antwoordspel van de adventure;
- Muziek gebruiken voor de geluidseffecten (+ openingstune?);
- Sproken gebruiken voor de animaties, de bewegende beelden.

In het boek zijn zeer veel voorbeelden te vinden van deze combinaties van 'werelden'.

Hoofdstuk 2

DE LOGO SCHILDPADWERELD



2.1 De schildpadwereld, een inleiding

De schildpadwereld, of de wereld van de Turtle Graphics, is wellicht voor de meeste mensen het meest herkenbare van Logo. De meeste mensen zullen ook door het zien van een of meerdere tekenende schildpadden voor het eerst in contact zijn gekomen met de programmeertaal Logo.

De wereld van de Turtle Graphics is, hoewel slechts één van de vier Logowerelden, toch wel zeer fascinerend. Met de Turtle Graphics, hoewel niet exclusief voor Logo, komen we in een wereld van lijnen, vormen en kleuren. Een wereld die zich kenmerkt door, wat Papert noemde, de 'Turtle Geometry'. Turtle Geometry betekent: schildpad-georiënteerde meetkunde. Wat dit inhoudt zal u hopelijk na het lezen van dit hoofdstuk duidelijk zijn.

Om de schildpadwereld te kunnen binnengaan zullen we de 'Turtle' moeten oproepen. We doen dat met de opdracht SHOW TURTLE, afgekort als ST. Na het intoetsen van deze twee letters, gevolgd door RETURN (of ENTER) zien we midden op het scherm een schildpad verschijnen.

Opmerking. Bij sommige Logo-versies verschijnt niet een vorm van een schildpad, maar een driehoekje.

Willen we de Turtle weer laten verdwijnen, dan is HT (HIDE TURTLE) voldoende.

Om te tekenen zullen we de Turtle in beweging moeten zetten. In principe kunnen we twee kanten op: vooruit (in Logo: FORWARD) of achteruit (in Logo: BACK).

De woorden FORWARD en BACK, primitieven geheten, vormen welhaast de belangrijkste opdrachten in de Turtle Graphics. Ook hier kunnen we afkortingen gebruiken, namelijk FD en BK.

Toetsen we nu FD en RETURN of ENTER in, dan doet de Turtle niets. Wel krijgen we de volgende boodschap in beeld:

```
NOT ENOUGH INPUTS TO FD (of FD NEEDS MORE INPUTS
                           of een Nederlandstalige melding)
```

Dat betekent dat Logo met alleen de opdracht FD niets kan doen; er moet nog iets bij, namelijk het aantal stappen dat vooruit gelopen moet worden. Bij de opdracht FD 100 (+ RETURN) zien we de Turtle razendsnel 100 stappen vooruitgaan. Hierbij laat hij een spoor achter zich. Zo kunnen we ook weer terug. Dat doen we met BK 100 (+ RETURN). Denk eraan dat na FD en BK een spatie moet staan. Begrijpt u nu waarom Logo een interactieve taal genoemd wordt?

De uitgangspositie van de Turtle, midden op het beeld wordt de thuispositie genoemd; deze duiden we in Logo aan met HOME. De Turtle staat dan met zijn neus verticaal naar boven (noordrichting). Waar de Turtle zich op het beeldscherm ook bevindt, steeds als we

HOME intoetsen zal hij terugkeren naar deze thuispositie waarbij hij een spoor achterlaat.

En nu de richting. Hoe kunnen we de Turtle laten draaien? Als we hem naar rechts willen draaien gebruiken we gewoon RIGHT (RT dus) en naar links LEFT (LT). Zouden we zomaar RT (+ RETURN) intoetsen dan krijgen we:

```
NOT ENOUGH INPUTS TO RT
```

Achter de woorden RT en LT moeten we een getal intoetsen, namelijk het aantal graden dat de Turtle moet draaien. Bij RT 90 draait de Turtle over 90° naar rechts, bij LT 270 draait hij 270° naar links, hetgeen hetzelfde resultaat geeft!

Zo zien we dat RT 90 hetzelfde resultaat geeft als LT 270 en RT 180 hetzelfde resultaat geeft als LT 180.

Om het scherm schoon te maken gebruiken we CS (Clear Screen) voor het grafische deel van het scherm, en CT (Clear Text) voor het tekstgedeelte.

Met de vier opdrachten FD, BK, RT en LT is bijna elke denkbare tekening te maken.

Als voorbeeld gaan we een vierkant maken. Dat doen we zo. Eerst laten we de Turtle 100 stappen vooruit (op het scherm verticaal naar boven) lopen met FD 100, daarna draaien we hem 90° rechtsom met RT 90. De looprichting is nu horizontaal naar rechts; we toetsen opnieuw FD 100. Om een vierkant te krijgen zullen we de Turtle nu opnieuw rechtsom moeten draaien, dus ... RT 90. Dit doen we nog twee keer. Uiteindelijk krijgen we dit:

```
FD 100 RT 90 FD 100 RT 90 FD 100 RT 90 FD 100 RT 90
```

Zo ontstaat een keurig vierkant.*)

Op zichzelf is deze manier van het gebruiken van Logo-opdrachten niet onjuist, maar wel heel erg omslachtig. Het blijkt namelijk dat we bij het tekenen van een vierkant twee opdrachten vier maal moeten herhalen, namelijk de opdrachten FD 100 en RT 90. De REPEAT-opdracht kan ons hierbij helpen.

REPEAT is een herhaalopdracht en in dit voorbeeld kunnen we hem zo gebruiken:

```
REPEAT 4 [FD 100 RT 90]
```

Denk erom: REPEAT moet altijd gevolgd worden door een spatie, een getal en de opdrachten die herhaald moeten worden. Deze opdrachten (het kan ook een procedure zijn) moeten altijd tussen vierkante haakjes ([en]) geschreven zijn.

---

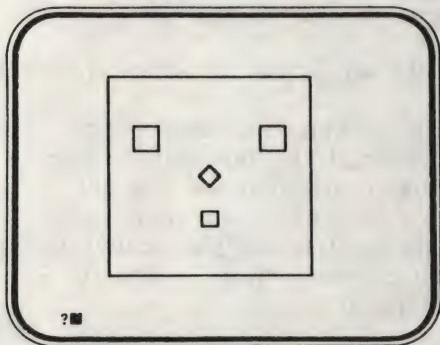
*) Bij veel computers is 100 stappen in verticale richting niet 'even lang' als 100 stappen in horizontale richting. We zien dan ook een keurige rechthoek in plaats van een keurig vierkant. Praktisch elke Logo beschikt over een opdracht (.ASPECT of .SETSCR) om deze resolutie bij te stellen. Zoek naar de juiste waarde voor uw systeem zodat het een echt vierkant wordt (kijk in uw handleiding).

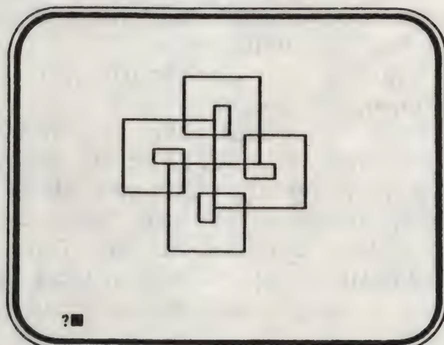
Opmerking. Denk erom dat Logo, in tegenstelling tot bijvoorbeeld BASIC, grote waarde hecht aan het juiste gebruik van spaties. FD50 en REPEAT4 worden door Logo niet herkend. Mocht dit toch eens gebeuren dan krijgen we de boodschap:

```
    I DON'T KNOW HOW TO REPEAT4
of: THERE'S NO PROCEDURE NAMED REPEAT4
```

Dit betekent dat Logo het woord REPEAT4 niet als opdracht (of als procedurenaam) herkent. Dat is gewoon een kwestie van wennen en zorgvuldig formuleren (en intoetsen).

Voordat we het maken van procedures gaan bespreken volgen nog enkele tekeningen; allemaal gemaakt met de opdrachten FD .., BK .., RT 90 en LT 90.





2.2 Procedures maken

Tot nu toe hebben we in Logo steeds gewerkt in de 'Direct Mode'. Dat wil zeggen: elke opdracht werd na 'RETURN' direct uitgevoerd. Maar Logo biedt de gebruiker ook de mogelijkheid om programma's te maken. In Logo noemen we een programma een procedure, zodat we deze term verder zullen gebruiken. Er zijn bijvoorbeeld procedures om een vierkant, een huis of een boot te tekenen. Maar ook procedures om muziek te maken, of een animatie of een taalspel.

Procedures maken doen we in de Editor. De Logo Editor roepen we op met EDIT "... of met TO ... . Achter het "-teken bij EDIT komt de procedurenaam te staan, want in Logo heeft iedere procedure een eigen naam. Pas op! Bij sommige Logo's moet het "-teken juist weggelaten worden!

Laten we als voorbeeld eens een procedure schrijven om een vier-

kant te tekenen. We zagen eerder al met welke opdrachten de Turtle een vierkant tekent. We zouden de procedure iedere naam kunnen geven, maar voor het gemak en de overzichtelijkheid noemen we de procedure gewoon VIERKANT. Dus toetsen we EDIT "VIERKANT (denk weer goed om de spaties) in. Als we vervolgens RETURN geven komen we in de Logo Editor. Op het scherm staat nu TO VIERKANT. Nu wacht Logo op de opdrachten die we in de procedure willen opnemen.

De opdrachten om een vierkant te tekenen kennen we inmiddels, zodat de procedure VIERKANT er zo uit komt te zien:

```
EDIT "VIERKANT
REPEAT 4 [FD 100 RT 90]
END
```

Elke procedure heeft END als laatste opdracht.



We verlaten de Logo Editor (bij Atari Logo) met de ESC-toets. Bij sommige Logo's zijn er andere methoden (bij BBC Logo bijvoorbeeld met de COPY-toets); kijk hiervoor in uw handleiding. We zijn uit de Logo Editor!

Logo laat zien dat de procedure VIERKANT in het werkgeheugen is opgeslagen door te melden:

```
VIERKANT DEFINED
```

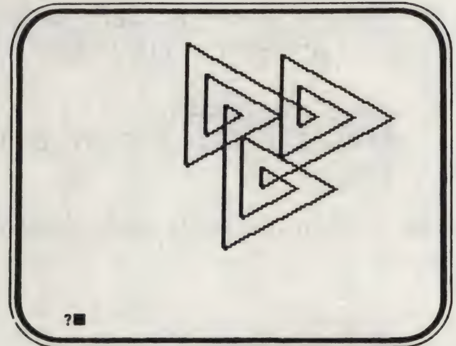
Maar hoe roepen we de procedure nu aan? Om de procedure te laten uitvoeren hoeven we hem alleen maar aan te roepen door het

intoetsen van de procedurenaam (+ RETURN), in dit geval VIERKANT. Op het moment dat we return geven treedt de procedure in werking en wordt een vierkant getekend.

En als we nu iets willen veranderen in de procedure? Dat is geen probleem, even opnieuw oproepen met EDIT "VIERKANT en ... daar is hij. Nu kan er het een en ander veranderd worden. Verlaat de Editor weer op de bekende manier.

Zijn er nog meer procedures?
Kijk maar eens:

```
EDIT "DRIEHOEK
REPEAT 3 [FD 100 RT 120]
END
```



en deze:

```
EDIT "CIRKEL
REPEAT 36 [FD 10 RT 10]
END
```

Met EDIT kunnen we elke denkbare procedure schrijven, iedere procedure met een eigen naam. Al deze procedures worden door Logo opgeslagen in de WORKSPACE, het werkgeheugen.

We kunnen de procedures aanroepen door het intoetsen van hun naam. We kunnen ze echter ook door een andere procedure laten aanroepen. Kijk maar eens:

```
EDIT "STER
REPEAT 36 [VIERKANT RT 10]
END
```

Als we deze procedure (STER dus) aanroepen door STER (+ RETURN) in te toetsen gebeurt het volgende. De Turtle tekent een vierkant (want de procedure VIERKANT hadden we al, nietwaar) en draait dan over 10 graden rechtsom. Dan tekent hij nog een vierkant en draait weer 10 graden rechtsom, en dan nog eens en nog eens en ..., in totaal 36 keer. Tenslotte komt de Turtle weer bij zijn beginpunt terecht, want 36 keer 10 graden is 360 graden (en dus een 'volledige draai').

We hebben de procedure VIERKANT gebruikt in een procedure STER. VIERKANT is nu de sub- en STER de hoofdprocedure. Dit kunnen we ook met de andere procedures doen, kijk maar:

```
EDIT "3DRIEHOEK
REPEAT 3 [DRIEHOEK RT 120]
END
```

of

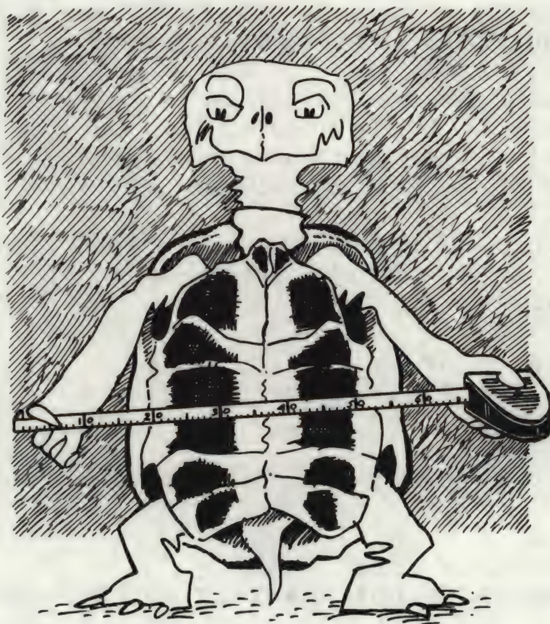
```
EDIT "6DRIEHOEK
REPEAT 6 [DRIEHOEK RT 60]
END
```

Natuurlijk kunnen we Logo ook voor ons laten rekenen. We zagen dat als we de Turtle één keer volledig rond willen laten draaien de totale draaihoek 360 graden moet zijn. Dit betekent dat het produkt van het aantal keren dat we de driehoek willen tekenen en de hoek waarover de driehoek moet draaien gelijk is aan 360 (3 × 120, 6 × 60). Deze regel passen we in het volgende voorbeeld toe:

```
EDIT "7DRIEHOEK
REPEAT 360/22 [DRIEHOEK RT 22]
END
```

Succes verzekerd! Logo rekent nu zelf het aantal keren uit dat de Turtle een driehoek moet tekenen.

2.3 *Variabelen*



De procedure VIERKANT is natuurlijk best wel aardig maar met deze procedure kunnen we alleen maar vierkanten tekenen met zijden van 100 stappen. Om met dezelfde procedure verschillende vierkanten

te kunnen tekenen moeten we het aantal stappen variabel maken. Kijk maar eens:

```
EDIT "VIERKANT :STAP
REPEAT 4 [FD :STAP RT 90]
END
```

Achter de procedurenaam hebben we nu een variabele gezet die STAP heet. Om Logo te laten zien dat het een variabele (en geen procedure) is zetten we er een dubbele punt voor, :STAP dus. De variabele :STAP wordt gebruikt om de Turtle te vertellen hoeveel stappen hij FORWARD (FD) moet gaan. Door in te toetsen VIERKANT 133 geven we de variabele :STAP de waarde 133. De Turtle zal nu de opdracht FD 133 uitvoeren (en natuurlijk de andere opdracht in VIERKANT). Als we echter VIERKANT 12.5 intoetsen krijgt de variabele :STAP de waarde 12.5, en daarmee vervalt zijn vorige waarde (dat was 133). Op deze manier kunnen we elk gewenst vierkant tekenen. En dat met slechts één procedure.

Toetsen we - per ongeluk - alleen VIERKANT in, dus zonder getal, dan zal Logo antwoorden met:

```
NOT ENOUGH INPUT TO VIERKANT
```

(of een andere foutmelding, afhankelijk van uw Logoversie)
Met andere woorden: de procedure VIERKANT kan zonder waarde voor de variabele :STAP niet worden uitgevoerd.

Opmerking. Het is niet noodzakelijk om een variabele een echt woord als naam (bijvoorbeeld STAP) te geven. Een variabele mag ook X heten of ZZ. Het is echter wel aan te raden de variabele te benoemen (zeker in het begin). Experimenteer maar eens wat met variabelen.

Als we een procedure als subprocedure willen gebruiken moeten we de eventueel daarin gebruikte variabelen ook bij de hoofdprocedure vermelden:

```
EDIT "STER :STAP
REPEAT 36 [VIERKANT :STAP RT 10]
END
```

Wilt u meer variabelen gebruiken? Dat is geen enkel probleem, kijk maar:

```
EDIT "SUPERSTER :STAP :HOEK
REPEAT 360/:HOEK [VIERKANT :STAP RT :HOEK]
END
```

Nu hebben we zowel de stapgrootte (door de variabele :STAP) als de hoekgrootte (door de variabele :HOEK) variabel gemaakt. Logo rekent dan zelf weer het aantal keren uit dat hij een vierkant

moet tekenen. Dit doet Logo door 360 (een volledige draaiing) te delen door de waarde van de variabele :HOEK. Heeft :HOEK de waarde 20 dan worden er 18 vierkanten getekend. Heeft :HOEK de waarde 3 dan worden er 120 vierkanten getekend. Dit geeft met slechts een paar procedures zeer veel mogelijkheden. We kunnen nu bijvoorbeeld intoetsen:

```
SUPERSTER 55 27.5
SUPERSTER 72.25 2
SUPERSTER 100 120
```

En steeds krijgen we een nieuwe figuur.

We kunnen echter de variabelen ook nog op een andere manier een waarde geven. Dat doen we dan met MAKE:

```
MAKE "STAP 100
```
(Denk erom niet :STAP)

Hiermee maken we een variabele die we STAP (denk aan de ") noemen, en we geven de variabele de waarde 100. Toetsen we nu in PRINT :STAP, dan is het resultaat 100. We gebruiken de aanhalingstekens dus alleen om een nieuwe variabele te definiëren. Is de variabele al bekend (en heeft deze al een waarde) dan gebruiken we de dubbele punt voor de naam van de variabele.

Goed, op dit moment heeft de variabele STAP de waarde 100. Kijk nu eens naar:

```
MAKE "STAP :STAP + 100
```

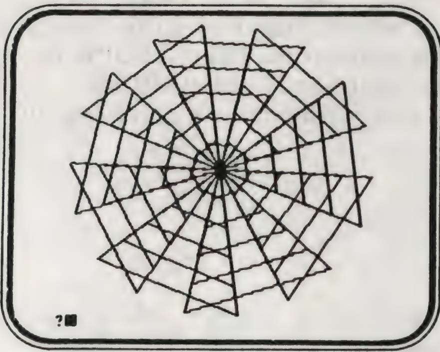
We maken een nieuwe variabele STAP, die de waarde heeft van de oude variabele :STAP (100) vermeerderd met 100. De variabele :STAP heeft nu dus de waarde 200.

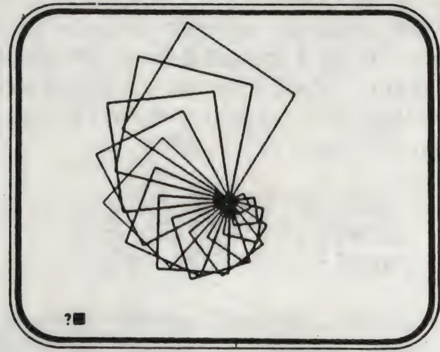
Dit kunnen we zo gebruiken:

```
EDIT "BLOKKENDOOS :STAP :KEER
REPEAT :KEER [VIERKANT :STAP MAKE "STAP :STAP + 10]
END
```

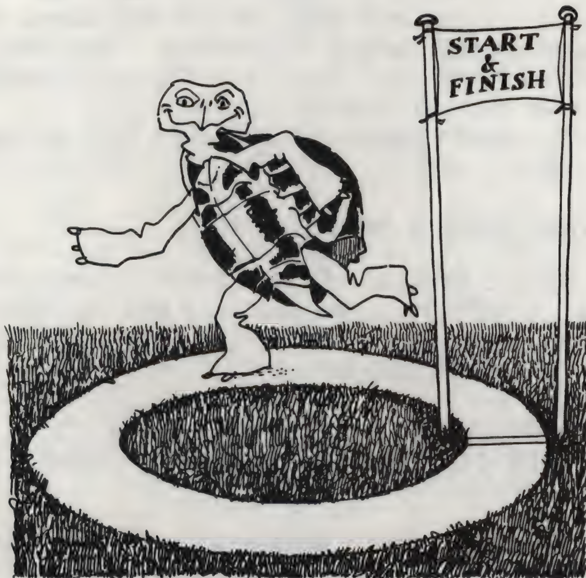
BLOKKENDOOS 20 5 zal dus 5 keer een vierkant tekenen. Eerst een vierkant met zijden van 20 stappen, dan een vierkant met zijden van 30 stappen, met zijden van 40 stappen, 50 stappen en tenslotte met zijden van 60 stappen. Dit komt doordat de variabele :STAP steeds met 10 verhoogd wordt.

We hoeven een variabele niet altijd een getal als waarde te geven. Wanneer we zeggen MAKE "NAAM LOGO, zal de variabele :NAAM, als waarde LOGO krijgen. Wanneer we dan intoetsen PRINT :NAAM zal LOGO worden afgedrukt. Je zou :NAAM een tekstvariabele kunnen noemen, maar hierover in de Logo-taalwereld meer.





2.4 Recursie



Iets bijzonders in de Logo Turtle Graphics is recursie. Laten we eerst eens kijken naar wat Seymour Papert zelf over recursie zegt:

"Recursie is een proces dat verwijst naar zichzelf. Omdat het naar zichzelf verwijst kan het ook naar zichzelf verwijzen terwijl het naar zichzelf verwijst. Om een vergelijking te maken: Het belangrijkste van denken is, dat je kunt denken over het denken. Je kunt zelfs denken over denken over denken. Logo heeft procedures die zichzelf aanroepen terwijl ze zichzelf aanroepen."

Duidelijk lijkt me ... of niet???

Kortom..., een recursieve procedure is een procedure die zichzelf aanroept. Dit kan eindeloos doorgaan, kijk maar eens naar dit voorbeeld:

```
EDIT "VIERKANT
REPEAT 4 [FD 100 RT 90]
VIERKANT
END
```

Omdat de procedurenaam in de procedure zelf is opgenomen zal de Turtle, wanneer hij de procedure uitvoert:

- eerst de opdracht REPEAT 4 [FD 100 RT 90] uitvoeren
- daarna roept de procedure als het ware zichzelf aan en begint opnieuw.

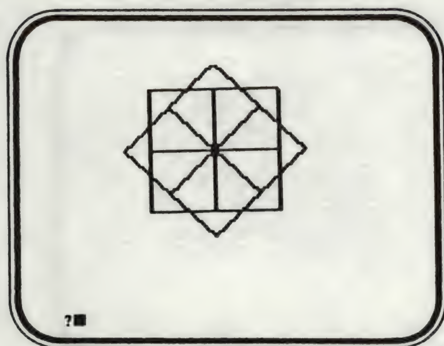
Het gevolg is dat de Turtle eindeloos de REPEAT-opdracht blijft herhalen. Steeds opnieuw wordt dus hetzelfde vierkant getekend. We kunnen de Turtle op een aantal manieren stoppen:

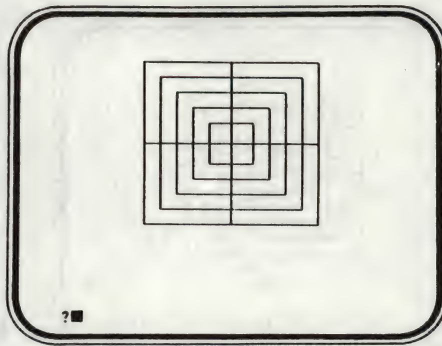
1. op de BREAK-toets drukken;
2. een 'SYSTEM RESET' uitvoeren;
3. de computer afzetten.

Allemaal radicale oplossingen en bovendien niet erg spectaculair. Gelukkig kunnen we dit ook anders oplossen. Kijk maar eens naar dit voorbeeld:

```
EDIT "VIERKANT :STAP
IF :STAP > 150 [STOP]
REPEAT 4 [FD :STAP RT 90]
VIERKANT :STAP + 5
END
```

Dit is een eenvoudige maar prima recursieve procedure.





Laten we deze recursieve procedure eens aan een nader onderzoek onderwerpen. We zullen hem regel voor regel bekijken.

- De procedure heet VIERKANT en heeft als variabele :STAP. We starten de procedure dus bijvoorbeeld met VIERKANT 5.
- IF :STAP >150 [STOP] is een testopdracht. Deze opdracht onderzoekt of de variabele :STAP een waarde heeft die groter is dan 150. Is dit het geval, dan stopt de procedure.
- REPEAT 4 [FD :STAP RT 90] is de opdracht voor het tekenen van een vierkant. Bij VIERKANT 5 is de waarde van de variabele :STAP dus 5 en zal de Turtle een vierkant tekenen met zijden van 5 stappen. Maar de waarde van :STAP verandert, dat zien we in de volgende regel:
- VIERKANT :STAP + 5; hier wordt de waarde van de variabele :STAP vermeerderd met 5. (De opdracht MAKE "STAP :STAP + 5 zou ook goed geweest zijn, maar zou dan binnen de procedure hebben moeten staan.) Deze laatste regel roept dus de procedure VIERKANT opnieuw aan. Alleen is de waarde van de variabele :STAP veranderd en wordt het vierkant dus groter.

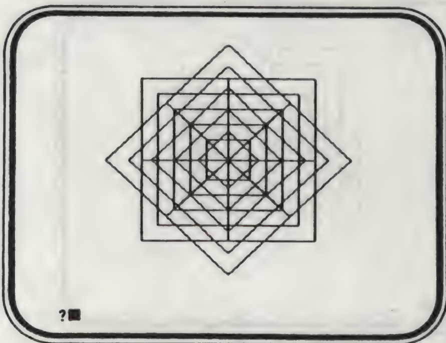
De procedure roept zichzelf net zolang aan totdat de variabele :STAP een waarde heeft die groter is dan 150.

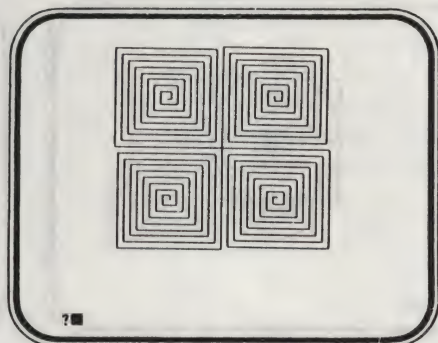
Natuurlijk kunnen we ook de vergrotingsfactor variabel maken. Dat is zelfs nog veel interessanter:

```
EDIT "VIERKANT :STAP :VERGROTING
IF :STAP > 150 [STOP]
REPEAT 4 [FD :STAP RT 90]
VIERKANT :STAP + :VERGROTING :VERGROTING
END
```

Vergeet niet om in de laatste regel (waar de procedure zichzelf aanroept) de variabele :VERGROTING te vermelden; anders zegt Logo iets als:

```
NOT ENOUGH INPUTS TO VIERKANT
```





Met deze en soortgelijke procedures kunnen we bijzonder aardige grafische figuren maken (zie figuren op de vorige bladzijde).

Maar er zijn meer manieren om het uitvoeren van een recursieve procedure te laten stoppen. We kunnen bijvoorbeeld een teller inbouwen, die bijhoudt hoe vaak de procedure zichzelf herhaalt. Kijk maar eens:

```
EDIT "FIGUUR :STAP :HOEK :TELLER
IF :TELLER < 1 [STOP]
FD :STAP RT :HOEK
MAKE "STAP :STAP + 10
FIGUUR :STAP :HOEK :TELLER - 1
END
```

Nu is de opdracht:

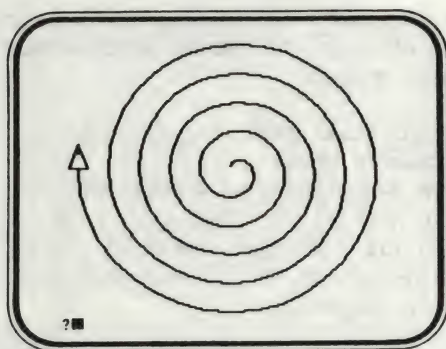
```
IF :TELLER < 1 [STOP]
```

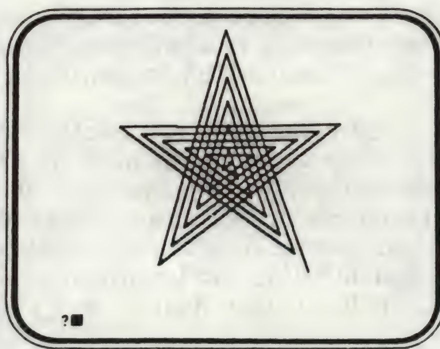
de testopdracht. Omdat de variabele :TELLER steeds met 1 afneemt zal de procedure uiteindelijk stoppen als TELLER kleiner dan 1 geworden is.

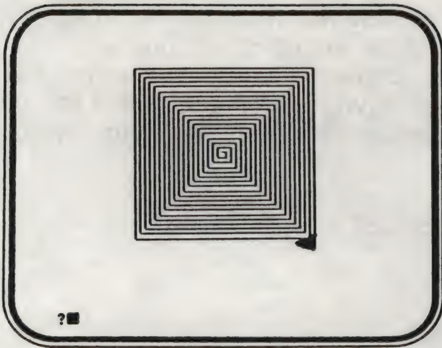
Het zal u opgevallen zijn dat zowel de stapgrootte als de hoek variabel zijn. We kunnen uit een en dezelfde procedure dan ook een veelheid van figuren halen. Bijvoorbeeld met:

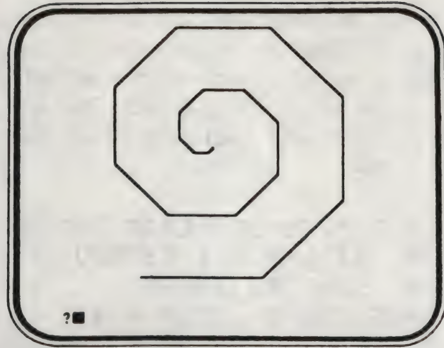
```
FIGUUR 10 90 50
FIGUUR 5 45 50
FIGUUR 10 120 50
enz.
```

Kijkt u maar eens naar de volgende voorbeelden. Ze zijn alle gemaakt met bovenstaande procedure:









Nog een manier om een recursieve procedure te stoppen is de volgende opdracht:

```
IF KEYP [STOP]
```

KEYP is niet een Logo-opdracht, maar een Logo-functie. KEYP kijkt of er ook een toets ingedrukt wordt. Is dit zo dat zal de procedure stoppen. Voor alle duidelijkheid: het doet er niet toe welke toets ingedrukt wordt.

Zullen we even een procedure met KEYP bekijken?

```
EDIT "FIGUUR :STAP :HOEK :VERGROTING
IF KEYP [STOP]
FD :STAP
RT :HOEK
FIGUUR :STAP + :VERGROTING :HOEK :VERGROTING
END
```

Tot zover deze toepassingen van recursie. Het is de moeite waard hier terdege mee te experimenteren. Alleen op die manier leert u hoe u de meest fantastische tekeningen kunt maken.

Bent u nu klaar voor een portie 'recursie voor gevorderden'? Laten we dan eens gaan kijken naar de recursieve boom.

De recursieve boom is een - binnen Logo-kringen - bekende procedure om te laten zien dat recursie niet per se 'tail-end' (aan het eind van de procedure) hoeft te staan. Recursie kan ook, meerdere malen zelfs, middenin een procedure staan. Kijk maar eens naar deze, op het eerste gezicht wat vreemde, procedure:

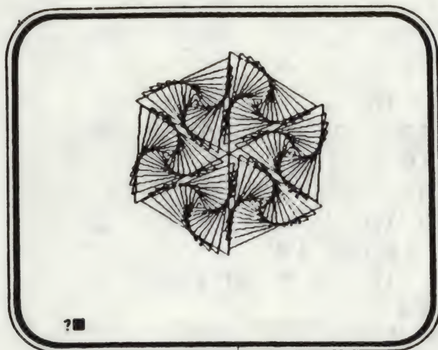
```
EDIT "BOOM :STAP
IF :STAP < 18 [STOP]
FD :STAP
RT 45
BOOM (3 * :STAP / 4)
LT 90
BOOM (3 * :STAP / 4)
RT 45
BK :STAP
END
```

Nu volgen enkele opmerkingen over deze procedure. We zullen hem eens stap voor stap volgen.

Opvallend is dat de procedure BOOM maar liefst drie keer wordt aangeroepen. We nemen als voorbeeld een stapgrootte van 32.

Opmerking. Met kleinere hoeken wordt de boom veel mooier! In plaats van RT 45 bijvoorbeeld RT 15 en in plaats van LT 90 bijvoorbeeld LT 30. Probeert u het maar eens.

Voor alle duidelijkheid wordt hieronder aangegeven wat er in de procedure gebeurt.



Bovenaan bladzijde 30 kunt u zien wat de Turtle doet als :STAP de waarde 32 heeft.

Let op de manier van inspringen. Dit geeft de verschillende procedureniveaus aan.

Niveau 1: Hier heeft :STAP de waarde van 32 en worden de opdrachten FD en RT uitgevoerd.

Niveau 2: Hier heeft :STAP de nieuwe waarde van 24 (3/4 van de oude waarde) en worden wederom de opdrachten FD en RT uitgevoerd.

Niveau 3: Hier heeft :STAP de nieuwe waarde van 18 (3/4 van de oude waarde) en worden de opdrachten FD en RT uitgevoerd.

```
BOOM 32
IF 32 < 18 [STOP]
FD 32
RT 45
BOOM 24 (= 3 MAAL 32 GEDEELD DOOR 4)
      BOOM 24
      IF 24 < 18 [STOP]
      FD 24
      RT 45
      BOOM 18 (= 3 MAAL 24 GEDEELD DOOR 4)
            BOOM 18
            IF 18 < 18 [STOP]
            FD 18
            RT 45
            BOOM 13 (= 3 MAAL 18 GEDEELD DOOR 4)
                  BOOM 13
                  IF 13 < 18 [STOP]
            LT 90
            BOOM 13
                  BOOM 13
                  IF 13 < 18 [STOP]
            RT 45
            BK 18
            END
      LT 90
      BOOM 18
            BOOM 18
            IF 18 < 18 [STOP]
            FD 18
            RT 45
            BOOM 13
                  BOOM 13
                  IF 13 < 18 [STOP]
            LT 90
            BOOM 13
                  BOOM 13
                  IF 13 < 18 [STOP]
            RT 45
            BK 18
            END
      RT 45
      BK 18
      END
enzovoort ..........
```

Niveau 4: Hier heeft :STAP de nieuwe waarde van 13 (3/4 van de oude waarde) en springt de procedure terug naar het derde niveau, omdat 13 kleiner is dan 18. Nu wordt echter de opdracht LT 90 uitgevoerd.

De test zorgt er dus voor dat, als de waarde van :STAP kleiner dan 18 wordt, de procedure naar een ander niveau springt.

Ook heel aardig is het om Logo zelf de waarde van een variabele in een recursieve procedure te laten bepalen. We doen dat met de RANDOM-functie, waarmee we toevalsgetallen kunnen oproepen.

In Logo werkt de RANDOM-functie als volgt:

RANDOM 10 geeft een van de volgende getallen: 0 1 2 3 4 5 6 7 8 of 9

(RANDOM 10) + 1 geeft een van de volgende getallen: 1 2 3 4 5 6 7 8 9 of 10

Om Logo zelf de waarde van een variabele te laten bepalen zeggen we bijvoorbeeld:

```
MAKE :STAP (RANDOM 10) + 5
```

De waarde van de variabele :STAP wordt zo een van de gehele getallen 5, 6, 7 ... of 14.

We geven u een voorbeeld van een drietal procedures die zorgen voor een 'levend schilderij'. Steeds wordt een willekeurige tekening gemaakt. Steeds wordt de waarde van de variabelen gewijzigd. Van tevoren kan aangegeven worden hoe vaak de hoofdprocedure zich moet herhalen. In totaal komen er vier variabelen in voor:

:S (= de stapgrootte [een RANDOM getal])
:H (= de hoekgrootte [een RANDOM getal])
:V (= de vergrotingsfactor [een RANDOM getal])
:T (= de teller ... van tevoren aangeven)

Hier zijn de procedures:

```
EDIT "SCHILDERIJ :T
IF KEYP [STOP]
ERN [:S :H :V]
KIES
TEKENING :S :H :V
WAIT 200 CS
SCHILDERIJ :T - 1
END

EDIT "KIES
MAKE "S (RANDOM 10) + 5
MAKE "H (RANDOM 90) + 45
MAKE "V (RANDOM 10) + 1
END

EDIT "TEKENING :S :H :V
IF  S > 150 [STOP]
FD :S RT :H
TEKENING :S + :V :H :V
END
```

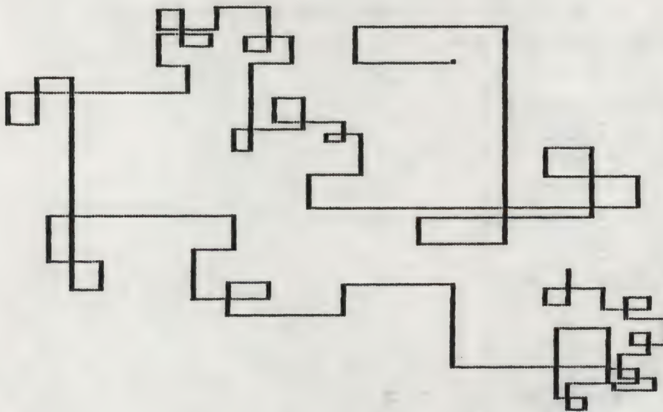
Op deze manier kunnen bijzonder aardige plaatjes ontstaan. Merk op dat met de opdracht ERN (ERASE NAME) de waarden van de variabelen worden gewist.

We kunnen de functie RANDOM ook in een procedure, die dezelfde structuur heeft als de procedure BOOM, gebruiken. Kijk maar:

```
EDIT "ZOMAARWAT :STAP
IF :STAP  < 10 [STOP]
FD :STAP
LT 1 + (RANDOM 3) * 90
ZOMAARWAT :STAP - 1
RT 90
ZOMAARWAT :STAP + 1
BK :STAP
END
```

De werking van deze ietwat vreemde recursieve procedure verschilt in wezen niet van die van de procedure BOOM.

Natuurlijk kunnen we zoiets krachtigs als recursie ook voor geheel andere dingen gebruiken (we komen er later bij Taal, Muziek en Sproken nog uitgebreid op terug).



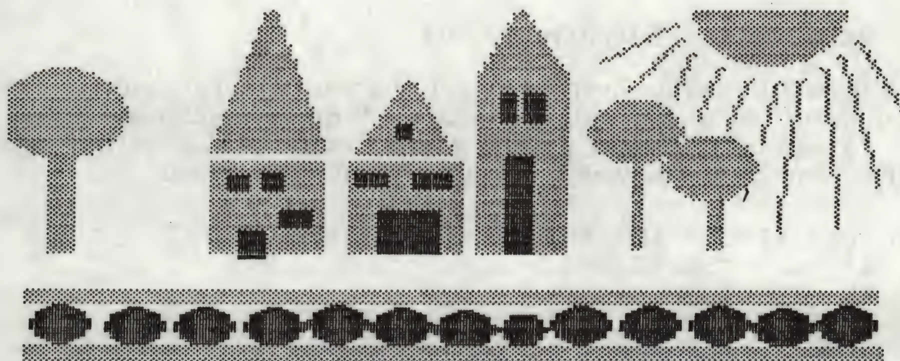
2.5 Logo pen- en kleuropdrachten



In deze paragraaf zullen we gaan kijken naar de verschillende pen-opdrachten en naar de uitgebreide kleuropdrachten.

Opmerking. Hier zijn tussen de verschillende Logo-versies aanzienlijke verschillen. Bekijk daarom zorgvuldig (indien u met een andere dan de LCSI Atari Logo werkt) de vergelijkingstabel achterin dit boek.

We zullen ook bekijken hoe we figuren kunnen inkleuren, zodat tekeningen als deze (door kinderen gemaakt) kunnen worden verkregen.



De Turtle beschikt over een tekenpen, waarmee hij zijn 'spoor' op het scherm kan trekken. Als we Logo starten, en de Turtle met ST (= Show Turtle) aanroepen, staat de tekenpen van de Turtle klaar om te tekenen. In Logo zeggen we PD (Pen Down). Met PU (Pen Up) laten we de Turtle als het ware zijn pen van het papier (scherm) optrekken, zodat zijn spoor onzichtbaar wordt. Kijk maar eens:

```
REPEAT 10 [PD FD 10 PU FD 10]
```

De Turtle tekent nu een keurige stippellijn.

Ook zijn de opdrachten PD en PU erg handig (en bijna onmisbaar) om verschillende figuren los van elkaar te tekenen. Maar er zijn meer opdrachten die de toestand van de pen veranderen:

PE (Pen Erase) en PX (Pen Reverse)

De opdracht PE maakt van de tekenpen een vlakgum. Kijk maar:

```
PD FD 100 WAIT 50 PE BK 100
```

Eerst wordt een lijn getekend en meteen daarna wordt deze lijn weer uitgegumd.

Opmerking. Met PE kan dus niet getekend worden!!!

Een wel heel bijzondere Logo-penopdracht is PX ofwel Pen Reverse. Eigenlijk zou deze opdracht PR moeten heten, maar PR staat al voor PRINT. PX is een penopdracht die zowel tekent als gumt, een soort combinatie van PD en PE dus. Tekenen waar nog niets staat, en gummen waar wel iets staat, dat kan fascinerende effecten geven. Kijk maar eens:

```
EDIT "VIERKANT :S
REPEAT 4 [FD :S RT 90]
END

PX REPEAT 3 [VIERKANT 100]
```

Ziet u wel? De eerste keer wordt een vierkant getekend, de tweede keer wordt het uitgegumd, en de derde keer weer getekend.

PX kan ook gebruikt worden voor 'animatie-achtige' effecten. Wat te denken van bijvoorbeeld een vierkant dat 'groeit' ...

```
EDIT "GROEIVIERKANT :S
IF :S > 150 [STOP]
PX REPEAT 2 [VIERKANT :S]
GROEIVIERKANT :S + 5
END
```

Opnieuw een recursieve procedure met een variabele :S (STAP) die een steeds grotere waarde krijgt. De opdracht IF :S > 150 [STOP] kijkt of de waarde van :S meer dan 150 is. Is dit zo, dan stopt de procedure.

Daarna wordt de opdracht Pen Reverse (PX) gegeven en maakt de Turtle tweemaal een vierkant: de eerste keer tekent hij en de tweede maal gumt hij. Daarna wordt de procedure opnieuw aangeroepen maar wordt de waarde van :S met 5 vermeerderd.

Natuurlijk kunnen we hiermee ook een figuur laten verplaatsen. Kijk maar:

```
PX HT REPEAT 10 [REPEAT 2 [VIERKANT 50] PU
    RT 90 FD 10 LT 90 PX]
```

Deze een-regelige-opdracht (one-liner) (geschreven zonder procedure in de DIRECT MODE) is even simpel als doeltreffend en illustreert op uitstekende wijze de mogelijkheden van een opdracht als PX. Nog een voorbeeld?

```
PX HT REPEAT 8 [REPEAT 2 [VIERKANT 30 ] RT 45]
```

En wat krijgen we??? Een vierkant dat om zijn eigen as draait.

Opmerking. Om een en ander wat sneller te maken is de Turtle onzichtbaar gemaakt met de opdracht HT (Hide Turtle).

Logo is niet alleen, wat de Turtle Graphics betreft, een wereld van figuren en vormen, maar ook zeker een wereld van kleuren. In LCSI Logo staan ons maar liefst zestien kleuren ter beschikking, terwijl iedere kleur ook nog eens acht tinten heeft.

Iedere kleur wordt aangeduid met een nummer, waarbij het laagste nummer de donkerste tint heeft. Dit is het volledige kleurenschema:

0 - 7	= grijs
8 - 25	= licht oranje
16 - 23	= oranje
24 - 31	= rood oranje
32 - 39	= rose
40 - 47	= paars
48 - 55	= paarsblauw
56 - 63	= blauw 1
64 - 71	= blauw 2
72 - 79	= licht blauw
80 - 87	= turquoise
88 - 95	= groen blauw
96 - 103	= groen
104 - 111	= geel groen
112 - 119	= oranje groen
120 - 127	= licht oranje

Willen we alle kleuren even bekijken? Dat kan met de volgende procedure, een handig middel om een mooie schermkleur uit te zoeken. Als de procedure is gestart kunnen we de beeldschermkleur veranderen door op een willekeurige toets te drukken. De procedure stopt als we S intoetsen. Het 'doorbladeren' van de kleurenkaart kan dus met iedere toets (met uitzondering van de S) gebeuren. Hier is de procedure:

```
EDIT "KLEURENKAART :N
SS HT SETBG :N CT
PR (SE [Het kleurnummer is :] :N
MAKE "KEUS RC
IF :KEUS = "S [STOP]
KLEURENKAART :N + 1
END
```

Met deze procedure kunnen we alle 127 kleuren van LCSI Logo bekijken, of een gedeelte daarvan. Zoeken we bijvoorbeeld een mooie kleur blauw, dan starten we de procedure met:

```
KLEURENKAART 48
```

en kunnen we van hieruit een geschikte kleur opzoeken.

Opmerking. Wellicht staan in de procedure KLEURENKAART nog een paar onbegrijpelijke opdrachten. Deze worden in het volgende hoofdstuk (over Logo Taal) nader besproken.

Er zijn drie dingen die we van kleur kunnen laten veranderen:

1. het beeldscherm;
2. de tekenpen(nen);
3. de Turtle(s).

1. De kleur van het beeldscherm veranderen we met de opdracht:

SETBG (gevolgd door een kleurnummer)
of: BG (van BackGround)

2. De kleur van de tekenpen(nen)

LCSI heeft de beschikking over drie tekenpennen met de nummers 0, 1 en 2. Als we Logo starten heeft de Turtle altijd pen 0 met als basiskleur 15. Pen 1 (te verkrijgen met de opdracht SETPN 1 = SETPenNumber) heeft als basiskleur 47. Pen 2 (te verkrijgen met de opdracht SETPN 2) heeft als basiskleur 121.

Het veranderen van de penkleur doen we met de opdracht:

SETPC :pen :kleur (soms PC van PenColour)

Vergeet niet om achter de opdracht SETPC te vermelden *welke* pen *welke* kleur moet krijgen.

3. De kleur van de Turtle(s)

Dat er meerdere Turtles zijn zullen we in de volgende paragraaf zien. Iedere Turtle kan een eigen kleur krijgen. We doen dat met:

SETC :kleur

Hoe we de verschillende Turtles aanspreken zullen we hierna zien.

Het inkleuren van vormen kan op verschillende manieren gebeuren. Steeds zullen we echter gebruik moeten maken van de coördinaten.

In Logo kunnen we gebruik maken van de opdrachten SETX en SETY:

- SETX voor het definiëren van X-coördinaten (horizontale Turtle positie)
- SETY voor het definiëren van Y-coördinaten (verticale Turtle positie)
- XCOR is een Logo-functie die Logo zelf laat uitzoeken wat de X-coördinaat van de Turtle is.
- YCOR is een Logo-functie die bepaalt wat de Y-coördinaat van de Turtle is.

SETX en SETY zijn opdrachten, waarop de Turtle reageert. XCOR en YCOR zijn functies, waaraan Logo zelf waarden toekent.

Zijn we met de Turtle aan het tekenen en vragen we: PR XCOR dan antwoordt Logo netjes met de juiste horizontale positie van de Turtle. Andere plaatsbepalende functies zijn:

XCOR, YCOR (geeft de X-positie en Y-positie van de Turtle)
HEADING (geeft de richting waarin de Turtle wijst; 0° is verticaal naar boven)

Vergeet niet om als de functiewaarden op het scherm afgedrukt moeten worden PR of PRINT (bijvoorbeeld PR XCOR) te gebruiken, anders kan Logo zoiets zeggen als:

```
YOU DON'T SAY WHAT TO DO WITH 0
```

We gaan eens kijken naar zo'n inkleurprocedure. Als voorbeeld nemen we weer een vierkant. We noemen de procedure KUBUS.

```
EDIT "KUBUS :S :K
SETPC 0 :K
REPEAT :S [FD :S SETX XCOR + 1 BK :S]
END
```

Natuurlijk hoeven we niet alleen de hoogte (:S) variabel te maken, dat kunnen we ook met de breedte doen. We maken zo de procedure BLOK:

```
EDIT "BLOK :S :B :K
SETPC 0 :K
REPEAT :B [FD :S SETX XCOR + 1 BK :S SETX XCOR + 1]
END
```

Wederom een procedure die even eenvoudig, als doeltreffend is. Door steeds de X-coördinaat met 1 op te hogen krijgen we als het ware allemaal lijnen naast elkaar. Zo ontstaat een ingekleurde figuur.

Er zijn echter nog andere manieren om figuren in te kleuren. Een ervan is deze:

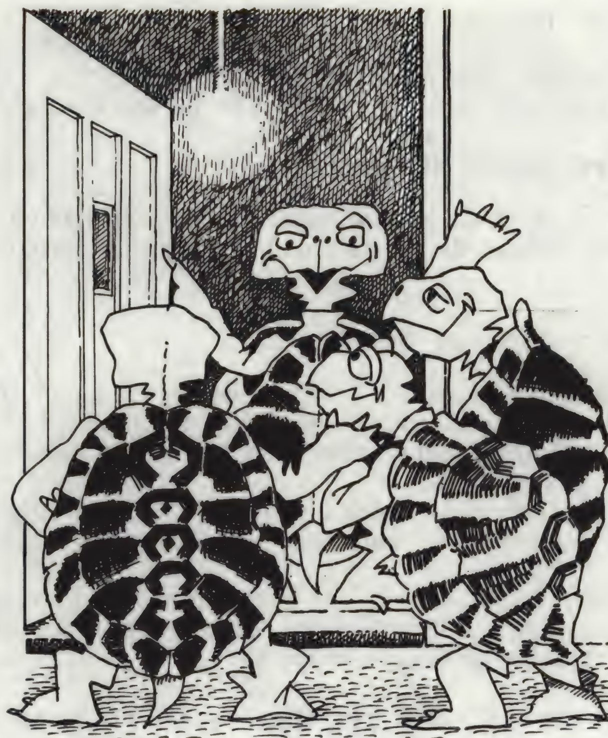
```
EDIT "KLEURIN :N
IF :N = "0 [STOP]
MAKE "XPOS XCOR
MAKE "YPOS YCOR
SETX :KPX SETY :KPY
SETX :XPOS SETY :YPOS
FD 1
KLEURIN :N - 1
END
```

```
EDIT "KP
MAKE "KPX XCOR
MAKE "KPY YCOR
END
```

Een waardevolle maar moeilijker procedure. Deze inkleurprocedure is pas te gebruiken wanneer we tijdens het tekenen een zogenaamd kleurpunt (KP) bepalen. Het inkleuren gebeurt namelijk doordat de Turtle steeds heen en weer loopt tussen zijn eigen coördinaten (die steeds veranderen; zie FD 1 in bovenstaande procedure) en het gekozen kleurpunt.

Willen we dus een vierkant inkleuren dan moeten we de Turtle eerst het kleurpunt laten opzoeken. Bij een vierkant is dat het rechterbovenhoekpunt (aangenomen dat de Turtle zich vlak voor het inkleuren in het linkerbenedenhoekpunt bevindt). Nu laten we de Turtle naar het kleurpunt gaan en leggen dit kleurpunt vast door het intoetsen van de procedurenaam KP. Daarna brengen we de Turtle terug naar het linkerbenedenhoekpunt van het vierkant. Pas dan kunnen we het vierkant inkleuren met KLEURIN :N. Voor :N geven we de lengte (aantal stappen) van de zijden van het vierkant op. Wij raden u aan zelf met dit inkleuren te experimenteren.

2.6 *Het werken met meerdere Turtles*



Een mooie mogelijkheid van LCSI Logo (en ook van vele andere Logo-versies) is het werken met meerdere Turtles tegelijk. In Atari LCSI Logo kan dat met maximaal vier Turtles. Zo kunnen we bijvoorbeeld meerdere Turtles tegelijk laten tekenen en we kunnen aardige animaties maken met meerdere figuren tegelijkertijd, maar dat komt later aan bod.

Als we Logo starten zien we na de opdracht ST (Show Turtle) een Turtle verschijnen. Deze Turtle zullen we nummer 0 noemen. Nu kunnen we met de opdracht:

```
TELL 1 ST
```

nog een Turtle oproepen. TELL 1 moet erbij om aan te geven welke Turtle we bedoelen.

Na deze opdracht zien we echter nog steeds maar één Turtle (nummer 0). Dat komt omdat de Turtles achter elkaar staan. Toetsen we FD 100 in, dan zal Turtle 1 (want dat was de laatst aangesproken Turtle) 100 stappen vooruit gaan, en blijft Turtle 0 rustig staan.

Nu roepen we de andere Turtles:

```
TELL 2 ST FD 75 (+ RETURN)
```

en

```
TELL 3 ST FD 50 (+ RETURN)
```

En daar zijn ze alle vier ...
Even nog Turtle 0 op gelijke afstand van de anderen zetten:

```
TELL 0 FD 25 (+ RETURN)
```

Zo, nu staan ze allemaal op gelijke afstand precies boven elkaar.

We kunnen echter de Turtles ook gezamenlijk aanspreken, kijk maar:

```
TELL [0 1 2 3] RT 90 FD 100
```

Alle vier de Turtles draaien nu 90° rechtsom en lopen 100 stappen vooruit. We hadden ook bijvoorbeeld met TELL [0 1] alleen de Turtles 0 en 1 kunnen aanspraken of alle Turtles behalve Turtle 2 met TELL [0 1 3].
Denk hierbij steeds om de vierkante haken [ ]; die zijn alleen nodig bij het tegelijk aanspreken van meerdere Turtles.

Alle Turtles voeren steeds de gegeven opdrachten tegelijk uit. Soms is het handiger de Turtles een voor een opdrachten te laten uitvoeren. Dat doen we bijvoorbeeld zo:

```
TELL [0 1 2 3] EACH [BK 200 RT 90 FD 50]
```

In dit geval zullen de opdrachten BK 200 RT 90 FD 50 door de Turtles om de beurt worden uitgevoerd. Dat komt door de opdracht EACH.

EACH wordt in de praktijk veel gebruikt samen met de functie WHO. Denk erom, WHO is geen opdracht maar een functie. De waarde van WHO is het cijfer dat correspondeert met de laatst aangesproken Turtle. Stel, we zijn aan het tekenen en toetsen dan PR WHO in. Komt er op het scherm te staan [0 1] dan weten we dat dit de laatst aangesproken Turtles zijn. Kijk maar eens naar de combinaties met EACH:

```
EACH [SETX WHO * 25]
EACH [SETC WHO * 8]
EACH [FD 50 * WHO]
EACH [RT 90 * WHO]
```

Bij de laatste regel: EACH [RT 90 * WHO] zal dus:

Turtle 0: 0 graden rechtsom draaien (90 * 0);
Turtle 1: 90 graden rechtsom draaien (90 * 1);

Turtle 2: 180 graden rechtsom draaien (90 * 2);
Turtle 3: 240 graden rechtsom draaien (90 * 3).

Zo zijn er veel aardige toepassingen te bedenken.

Tot slot nog de hulpprocedure USE. De procedure USE is gemaakt om de volgende reeks omslachtige handelingen te vermijden. Stelt u zich eens voor:

- we werken met vier Turtles tegelijk;
- daarvoor hebben we eerst de opdracht TELL [0 1 2 3] gegeven;
- nu willen we, laten we zeggen, Turtle 1 iets anders laten doen;
- dan moeten we hem eerst aanspreken;
- dat doen we met TELL 1 (gevolgd door de opdrachten die hij moet uitvoeren;
- daarna moeten we de overige Turtles weer aanspreken om verder te gaan;
- dus TELL [0 2 3] (gevolgd door hun opdrachten).

Allemaal ontzettend omslachtig (en in de praktijk komt een situatie als deze vaak voor), maar gelukkig is er USE.

USE kunnen we gebruiken om een Turtle aan te spreken zonder daarbij de anderen 'lastig te vallen'. We kunnen dan bijvoorbeeld zeggen USE 0 [RT 90] zonder de TELL-opdracht steeds te wijzigen. Hier is de procedure:

```
EDIT "USE :turtle :opdracht
MAKE "in.actie WHO
TELL :turtle
RUN :opdracht
TELL :in.actie
ERN "in.actie
END
```

Opmerking. Het is voor het gebruik handiger om de variabelen :turtle, :in.actie en :opdracht af te korten. Voor alle duidelijkheid hebben we ze nu laten staan.

Over de opdracht RUN hadden we het nog niet gehad. Met RUN laten we Logo, in een procedure, één of meer opdrachten uitvoeren alsof ze direct waren ingetypt. De invoer voor de RUN-opdracht moet in de vorm van een lijst gegeven worden. Bijvoorbeeld:

```
RUN [FD 50 RT 90]
```

Opmerking. De invoer voor RUN kan zowel een opdracht als een functie zijn. Indien de invoer van RUN een functie is, is de uitvoer van RUN gelijk aan die van de gebruikte functie. Bijvoorbeeld:

de uitvoer van: SUM 10 20

is gelijk aan de uitvoer van: RUN [SUM 10 20]

Tot besluit

Enkele Logo-promitieven betreffende de Turtle Graphics hebben we nog niet behandeld. En toch zijn ze niet onbelangrijk. U zult ze in veel procedures tegenkomen. Dit zijn er enkele:

SETPOS [xcoord. ycoord.] verplaatst de Turtle (al tekenend) naar de genoemde positie.
SETH 240 draait de Turtle rechtsom over het aangegeven aantal graden.

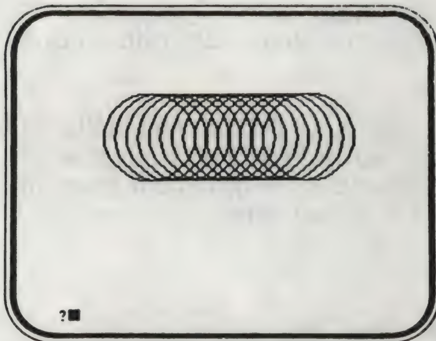
Tenslotte nog enkele tekenprocedures. Deze zorgen voor een aardige collectie grafische afbeeldingen. Ze worden gegeven zonder verder commentaar. Het advies is: gewoon proberen.

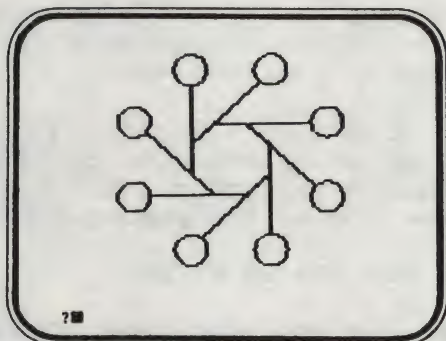
Het wegschrijven naar cassette-recorder, disk-drive of printer gaat met:

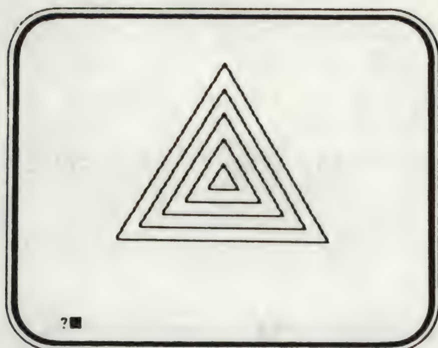
SAVE "C:	voor het opslaan op tape
SAVE "D:filenaam	voor het opslaan op disk
SAVE "P:	voor het afdrukken van de procedures

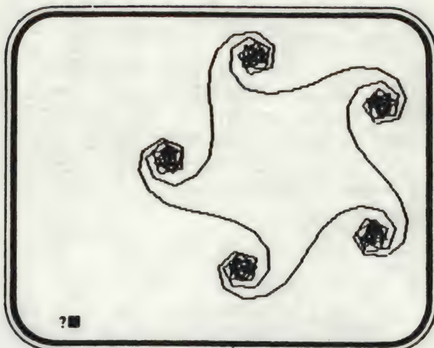
Voor het laden gebruiken we LOAD in plaats van SAVE. Hier volgen de procedures:

```
EDIT "POLYGOON :ZIJDE :HOEK
REPEAT 360 / :HOEK [FD :ZIJDE RT :HOEK]
IF REMAINDER 360 :HOEK = "0 [STOP]
POLYGOON :ZIJDE :HOEK
END
```










```
TO VERPLAATS :vergroting
PU BK :vergroting / 2 LT 90
FD :vergroting / 2 RT 90 PD
END

TO VIERKANT :stap
REPEAT 4 [FD :stap RT 90]
END

TO VEELVIERKANT :stap :vergroting
IF :stap > "150 [STOP CS]
VIERKANT :stap
VERPLAATS :vergroting
VEELVIERKANT :stap + :vergroting :vergroting
END

TO VEELVIERKANTEN :keer
CS FS SETBG 0
MAKE "stap ( RANDOM 25 ) + 1
MAKE "vergroting ( RANDOM 20 ) + 1.5
IF :keer = "0 [SS STOP]
VEELVIERKANT :stap :vergroting
ERN "stap ERN "vergroting
PU HOME CS PD
VEELVIERKANTEN :keer - 1
END


TO DUBBELSPIRAAL :teller
FS CS HOME
MAKE "pen1 ( RANDOM 8 ) * ( RANDOM 17 )
MAKE "pen2 :pen1 + 2
TELL 0 ST SETPC 0 :pen1 SETPN 0
TELL 1 ST SETX 5 SETPC 1 :pen2 SETPN 1
TELL [0 1] PD
IF :teller = "0 [SS STOP]
MAKE "stap ( RANDOM 10 ) + 1
MAKE "vergroting ( RANDOM 5 ) + 1
MAKE "hoek ( RANDOM 46 ) + 45
REPEAT 60 [FD :stap RT :hoek MAKE "stap :stap + :vergroting]
DUBBELSPIRAAL :teller - 1
END
```

```
TO SPIRAAL :teller
FS CS HOME
IF :teller = "0 [SS STOP]
MAKE "stap ( RANDOM 10 ) + 1
MAKE "vergroting ( RANDOM 5 ) + 1
MAKE "hoek ( RANDOM 46 ) + 45
REPEAT 60 [FD :stap RT :hoek MAKE "stap :stap + :vergroting]
SPIRAAL :teller - 1
END


TO POLYGOON2
MAKE "stap ( RANDOM 50 ) + 20
REPEAT 10 [RT 36 REPEAT 5 [FD :stap RT 72]]
END
```

Hoofdstuk 3 – **DE LOGO TAALWERELD**



3.1 Inleiding

De Logo-taalwereld is een geheel andere wereld, of leeromgeving, dan die van de Turtle Graphics. Aan de Logo-taalwereld kunnen we zien dat Logo nauw verwant is met talen die gebruikt worden op het gebied van kunstmatige intelligentie, zoals bijvoorbeeld LISP en Prolog.

De subtitel van dit hoofdstuk is: het manipuleren met woorden en lijsten. Om dit te begrijpen moeten we eerst met elkaar afspreken wat we hiermee precies bedoelen.

Een woord in Logo bestaat uit tekens, net zoals in onze Nederlandse taal. Alleen is het begrip WOORD in Logo veel omvangrijker. Zo zijn voor Logo TAFEL, STOEL, AANHANGWAGEN of TENTENTENTOONSTELLING woorden, maar ook 1234, BOEM8, 9%tr2RRR en ??.

Opmerking. We praten hier over de definitie van het Logo-begrip WOORD, en niet over de betekenis van woorden.

Kortom, een WOORD is een verzameling tekens, al dan niet met een uniforme of individuele betekenis.

Een LIJST (in Logo) is een groep woorden of een groep tekens. Een lijst wordt altijd tussen [vierkante haken] geschreven. Dit zijn enkele lijsten:

[KAT HOND KIP] is een lijst die drie elementen bevat;
[PRINT [Hallo allemaal]] is een lijst die twee elementen bevat.

U ziet het: als we twee woorden als één element willen interpreteren, moeten deze twee woorden omsloten zijn door [ ].

Het bovenstaande is uitermate belangrijk. Zeker wanneer we belangrijke informatie in een lijst stoppen. Dit is bijvoorbeeld het geval wanneer we een gegevensbestand in Logo gaan maken. Kijk maar eens naar deze lijst:

[[L.C.S.I.Logo] [Turtlelaan 33] [Recursia] [Tel.0001-987654]]

Deze lijst bevat vier elementen:
1. de naam
2. het adres
3. de woonplaats
4. het telefoonnummer.

Door nu te manipuleren met woorden en lijsten kunnen we bijvoorbeeld
- alle namen (eerste elementen) laten zien (of printen);
- sorteren op naam, adres, woonplaats, etc.

Kortom, we kunnen allerlei bewerkingen uitvoeren op de gegevens die in een gegevensbestand zitten. Later komen we op het gegevens-

bestand terug. Het werd op dit moment genoemd om de begrippen WOORD, ELEMENT en LIJST te verduidelijken.

3.2 Het manipuleren met woorden en lijsten



In deze paragraaf leren we er weer veel nieuwe woorden bij. Er wordt geprobeerd steeds bij ieder woord voorbeelden te geven. Het bekendste woord in deze Logo-wereld is:

PRINT (of PR)

Toets in:

PR Hallo	resultaat: I DONT'T KNOW HOW TO Hallo
PR "Hallo	resultaat: Hallo
PR [Hallo Allemaal]	resultaat: Hallo Allemaal
PR ["Hallo "Allemaal]	resultaat: "Hallo "Allemaal
PR "	resultaat:
PR []	resultaat:

Dit zijn de gewone print-opdrachten. Nu gaan we eens kijken naar opdrachten om met woorden en lijsten te manipuleren.
Toets in:

```
PR FIRST "LOGOBOEK          resultaat: L

PR FIRST [LOGO BOEK]        resultaat: LOGO

PR LAST  "LOGOBOEK          resultaat: K

PR LAST  [LOGO BOEK]        resultaat: BOEK
```

Maar er is meer:

```
PR BF "LOGOBOEK             resultaat: OGOBOEK

PR BF [LOGO BOEK]           resultaat: BOEK

PR BL "LOGOBOEK             resultaat: LOGOBOE

PR BL [LOGO BOEK]           resultaat: LOGO
```

Zoals u waarschijnlijk al hebt begrepen staat BL voor BUTLAST en BF voor BUTFIRST.

BL geeft als resultaat alle tekens of woorden behalve het laatste.
BF geeft als resultaat alle tekens of woorden behalve het eerste.

BF, FIRST, BL en LAST kunnen afzonderlijk gebruikt worden, maar er zijn ook combinaties mogelijk. Kijk maar eens naar deze voorbeelden:

```
PR FIRST BF "Computer
= o

PR FIRST BF [Logo Taal Computer]
= Taal

PR LAST BL "Computer
= e

PR LAST BL [Logo Taal Computer]
= Taal
```

Hier zijn heel aardige dingen mee te doen.
Voordat we straks naar de serieuze taalwereld gaan, laten we eerst nog enkele woordgrapjes zien. Kijk eens naar deze procedures:

```
EDIT "ERAF1 :WOORD
IF EMPTYP :WOORD [STOP]
PR :WOORD
ERAF1 BF :WOORD
END
```

Deze procedure (die een woord als invoer vraagt) heeft het volgende resultaat:

```
?ERAF1 "COMPUTER
COMPUTER
OMPUTER
MPUTER
PUTER
UTER
TER
ER
R
?
```

Aan de achterkant beginnen kan ook:

```
EDIT "ERAF2 :WOORD
IF EMPTYP :WOORD [STOP]
PR :WOORD
ERAF2 BL :WOORD
END
```

Nu wordt het resultaat:

```
?ERAF2 "COMPUTER
COMPUTER
COMPUTE
COMPUT
COMPU
COMP
COM
CO
C
?
```

Dit kan echter ook met zinnen. Dan maken we van de variabele :WOORD een lijst. Kijk maar:

```
?ERAF2 [Een computer is op zich een dom apparaat]
Een computer is op zich een dom apparaat
Een computer is op zich een dom
Een computer is op zich een
Een computer is op zich
Een computer is op
Een computer is
Een computer
Een
?
```

De opdracht : IF EMPTYP :WOORD [STOP] is weer zo'n testopdracht waar we al eerder over spraken. De opdracht controleert iedere keer dat de recursieve procedure doorlopen wordt of de variabele :WOORD nog elementen of woorden bevat. Zo niet dan is de variabele leeg (EMPTY) en stopt de procedure.

De volgende procedure lijkt hierop:

```
EDIT "ACHTERUIT :INVOER
IF EMPTYP :INVOER [STOP]
TYPE LAST :INVOER
TYPE "
ACHTERUIT BL :INVOER
END
```

En kijk eens naar het resultaat:

```
?ACHTERUIT "Computer

r e t u p m o C ?
```

of:

```
?ACHTERUIT [Ik lees een boek over Logo]

Logo over boek een lees Ik ?
```

Let op. In deze procedure is de Logo-opdracht TYPE gebruikt in plaats van PRINT om de letters of woorden naast elkaar te laten afdrukken. Als we PRINT gebruiken worden alle letters of alle woorden niet naast maar onder elkaar afgedrukt. De opdracht TYPE zal ook in latere procedures nog veelvuldig worden toegepast.

Overigens geeft ook TYPE " een spatie, net zoals PR ".

Tot nu toe hebben we ons steeds beziggehouden met het uit elkaar halen van woorden en lijsten. Er zijn echter ook Logo-opdrachten voor het samenvoegen van woorden en lijsten. We zullen ze een voor een bekijken. Tevens is bij iedere opdracht een voorbeeldprocedure opgenomen.

Het gaat om de volgende opdrachten:

```
WORD
LIST
COUNT
LPUT
FPUT
SE
```

WORD

De Logo-functie WORD voegt twee of meer woorden tezamen:

```
PR WORD "Logo "Computer
LogoComputer

PR (WORD "Logo "Computer "Boek)
LogoComputerBoek
```

U ziet dat, wanneer we meer dan twee woorden of elementen willen samenvoegen, deze woorden of elementen tussen haakjes gezet moeten worden.

Een voorbeeldje. Sommige mensen hebben de neiging om in hun manier van praten nogal vaak het woordje 'ja' te gebruiken. Deze eenvoudige procodure doet dat ook:

```
EDIT "PLUSJA :LIJST
IF EMPTYP :LIJST [STOP]
PR WORD FIRST :LIJST "ja
PLUSJA BF :LIJST
END
```

Kijk maar:

```
PLUSJA [Ik heb er ook geen verstand van]
Ikja hebja erja ookja geenja verstandja vanja
```

LIST

De werking van deze Logo-functie lijkt erg veel op die van WORD. LIST maakt echter van twee elementen niet een woord, maar een lijst. Kijk maar:

```
LIST "LOGO "BOEK
[LOGO BOEK]
```

We zagen al eerder dat er ook lijsten in lijsten bestaan:

```
LIST "LOGO [BOEK]
[LOGO [BOEK]]

LIST [LOGO] [LOGO BOEK]
[[LOGO] [LOGO BOEK]]
```

Ook nu moeten, wanneer we meer dan twee elementen/lijsten willen samenvoegen, deze elementen/lijsten tussen haakjes staan:

```
(LIST [Naam] [Adres] [Woonplaats])
[[Naam] [Adres] [Woonplaats]]
```

COUNT

De functie COUNT is gewoon een teller. COUNT telt het aantal elementen van een lijst of een woord:

```
PR COUNT "Logo
4

PR COUNT [Basic [Logo Lisp] Prolog]
3
```

Met COUNT kunnen we heel zinvolle dingen doen. Zo kunnen we bijvoorbeeld de procedure ITEM maken. In sommige Logo-versies is ITEM een Logo-opdracht, maar als hij niet in uw Logo-versie is opgenomen, maakt u hem toch zelf?

```
EDIT "ITEM :N :OBJECT
IF EMPTYP :OBJECT [OP "]
IF :N = 1 [OP FIRST :OBJECT
OP ITEM :N - 1 BF :OBJECT
END
```

Zo geeft `PR ITEM 2 [Een Logo Computer]` : `Logo`
en `PR ITEM 6 "Logoprocedure` : `r`

ITEM vraagt dus om twee invoergegevens:
1. het hoeveelste teken/element;
2. van welk woord/welke lijst.

Een procedure op basis van ITEM die we later veel zullen gebruiken en die ook in uw eigen procedures heel zinvol kan zijn is KIES. KIES haalt een willekeurig element uit een lijst (of een willekeurig teken uit een woord).

De (mini)procedure KIES is als volgt:

```
EDIT "KIES :LIJST
PR ITEM (1 + RANDOM COUNT :LIJST) :LIJST
END
```

Let even goed op 1 + RANDOM COUNT. Wanneer een lijst bijvoorbeeld zes elementen heeft, dan is de waarde van COUNT 6. RANDOM 6 kan echter slechts 0,1,2,3,4,5 als waarde geven. Een ITEM 0 staat niet in de lijst en ITEM 6 (het laatste element van de lijst) zullen we nooit krijgen. Daarom tellen we bij RANDOM COUNT 1 op. Het resultaat is een volkomen willekeurig element van een lijst:

`KIES [een twee drie vier vijf]` geeft bijvoorbeeld : `vier`

Als we de programma's PSYCHIATER en HAIKU.GENERATOR behandelen, zult u zien hoe zinvol deze procedures zijn.

LPUT en FPUT

Als we een nieuw element aan een lijst willen toevoegen, kunnen we gebruik maken van de opdracht LPUT of FPUT:
LPUT (Last PUT) zet het nieuwe element achter in de lijst;
FPUT (First PUT) zet het nieuwe element voor in de lijst:

```
PR LIST LPUT "laatst [eerst]
[eerst laatst]

PR LIST FPUT "laatst [eerst]
[laatst eerst]
```

SE

SE (SEntence) wordt gebruikt om woorden, lijsten of een combinatie van beide met elkaar te verbinden:

```
PR SE [DIT ZIJN][TWEE LIJSTEN]
DIT ZIJN TWEE LIJSTEN

PR SE "TWEE "WOORDEN
TWEE WOORDEN
```

In de praktijk maken we vaak gebruik van de opdracht SE om woorden of zinnen te verbinden met variabelen:

```
MAKE "NAAM [Henk Jan] PR SE [Hoe gaat het ermee ] :NAAM
Hoe gaat het ermee Henk Jan
```

of:

```
MAKE "LEEFTIJD 30 PR (SE [Ben jij al] :LEEFTIJD "jaar "?)
Ben jij al 30 jaar ?
```

Ook nu zien we weer dat, wanneer SE een invoer heeft van meer dan twee woorden (of lijsten), we haakjes moeten gebruiken. (In dit geval kunnen we volstaan met één stel.)

Voordat we de opdrachten WORD, LIST, COUNT, LPUT, FPUT en SE aan de hand van enkele programma's nader gaan bekijken, eerst nog even aandacht voor het volgende.

We zijn al verschillende malen in procedures zogenaamde testopdrachten tegengekomen, bijvoorbeeld een testopdracht die kijkt of een lijst nog niet 'leeg' is. Logo heeft verschillende woorden die in testopdrachten goed te gebruiken zijn. Deze woorden, Logo-functies genoemd, zijn geen opdrachten; ze kijken alleen of iets waar of onwaar is. De waarde van deze woorden is steeds: TRUE of FALSE. Aan deze waarde kunnen we binnen de testopdracht echter wel een handeling verbinden. Kijk maar:

```
IF EMPTYP :LIJST [STOP]
```

Nu zal EMPTYP kijken of de variabele :LIJST al leeg is. De waarde van EMPTYP zal:
- als de lijst NIET leeg is FALSE zijn;
- als de lijst WEL leeg is TRUE zijn.
En als de waarde van EMPTYP TRUE is, wordt de opdracht [STOP] uitgevoerd.

3.3 Het maken van testopdrachten en de te gebruiken Logo-functies

We zagen al hoe we EMPTYP konden gebruiken in een testopdracht. Er zijn echter nog meer Logo-woorden. Elk van deze woorden kijkt naar een ander aspect van de invoer. Hieronder staan enkele van deze Logo-woorden; u zult ze later in de verschillende procedures nog regelmatig tegenkomen:

```
NAMEP
WORDP
NUMBERP
LISTP
MEMBERP
EQUALP
```

Nu volgen enkele voorbeelden van het gebruik van deze woorden.

- NAMEP kijkt of een variabele wel een waarde heeft. Een voorbeeld:

```
PR NAMEP "LIJST
FALSE

MAKE "LIJST [Dit is een lijst]  PR NAMEP "LIJST
TRUE
```

In het eerste geval heeft de variabele :LIJST geen tekstwaarde (en daarom heeft de functie NAMEP de waarde False). Daarna hebben we :LIJST de waarde [Dit is een lijst] gegeven.
In een testopdracht gebruiken we NAMEP als volgt:

```
IF NAMEP :LIJST [PR [Deze variabele heeft een waarde]]
                [PR [Geen waarde]]
```

Hier zien we dat, indien de waarde van NAMEP TRUE is, de eerste opdracht wordt uitgevoerd. Indien de waarde van NAMEP FALSE is, wordt de tweede opdracht uitgevoerd.

- WORDP kijkt of hetgeen ingevoerd is wel een woord is. Bijvoorbeeld:

```
PR WORDP "ABCDEFG'
TRUE

PR WORDP [Lijst]
FALSE
```

In het eerste geval valt ABCDEFG volgens de normen van Logo inderdaad onder de definitie van woord. [Lijst] echter is niet een woord maar een lijst met één element.
Ook WORDP kunnen we gebruiken in een testopdracht:

```
IF WORDP :LIJST [PR [Dit is een woord]]
                [PR [Dit is geen woord]]
```

- NUMBERP kijkt of de invoer misschien een getal is. Bijvoorbeeld:

```
PR NUMBERP 7
TRUE

PR NUMBERP [9]
FALSE
```

Of als testopdracht:

```
IF NUMBERP :GETAL [MAKE "GETAL :GETAL + 1]
                  [PR [Dit is geen getal]]
```

- LISTP kijkt of de invoer wellicht een lijst is. Bijvoorbeeld:

```
PR LISTP "HALLO
FALSE

PR LISTP [HALLO]
TRUE
```

En ook in testopdrachten kan LISTP gebruikt worden:

```
IF LISTP :GROET [PR COUNT :GROET]
                [PR [Dit is geen lijst]]
```

Van een iets ander kaliber, maar ook zeer bruikbaar is MEMBERP.

- MEMBERP is een zeer interessante Logo-functie, die twee invoergegevens vraagt:
  1. een getal, letter, woord of lijst;
  2. een lijst.

MEMBERP kijkt of het eerste invoergegeven een element van het tweede invoergegeven is. Bijvoorbeeld:

```
PR MEMBERP A [A B C D E]
TRUE

PR MEMBERP C [A B [C] D E]
FALSE
```

Om de betekenis van MEMBERP te illustreren laten we de volgende mini-procedure zien. Deze procedure KLINKER kijkt of een woord met een klinker begint en maakt daarbij gebruik van MEMBERP:

```
EDIT "KLINKER :WOORD
OUTPUT MEMBERP FIRST :WOORD [A E I O U]
END
```

- EQUALP tenslotte is een vergelijkingsfunctie en wordt gebruikt bij letters, getallen, woorden en lijsten, bijvoorbeeld:

```
PR EQUALP 150 50 * 3
TRUE

PR EQUALP " []
FALSE
```

Let op!! Een leeg woord " is NIET hetzelfde als een lege lijst [ ].

EQUALP kunnen we bijvoorbeeld gebruiken om te zien waar een element zich bevindt in een lijst:

```
EDIT "WAAR? :ELEMENT :LIJST
IF EMPTYP :LIJST [OUTPUT 0]
IF EQUALP :ELEMENT LAST :LIJST [OUTPUT COUNT :LIJST]
OUTPUT  WAAR? :ELEMENT BUTLAST :LIJST
END
```

Kijk maar:

```
PR WAAR? "3 [9 8 7 6 5 4 3 2 1]
7

PR WAAR? "n "computerboeken
14
```

Nog enkele bijzondere testopdrachten, die we aan de hand van een aantal voorbeelden zullen verduidelijken:

- AND

```
EDIT "TEST1 :W
IF AND FIRST :W = "A LAST :W = "A [PR [Dit woord
  begint en eindigt met een A][Volgende woord ..]
END
```

Kijk maar:

```
TEST1 "ACHTERNA
Dit woord begint en eindigt met een A

TEST1 "Achterna
Volgende woord ..
```

Hier wordt de eerste printopdracht

```
Dit woord begint en eindigt met een A
```

pas uitgevoerd als aan beide voorwaarden is voldaan. De voorwaarden zijn:

- de eerste letter van het in te voeren woord moet een A zijn;
- de laatste letter van het in te voeren woord moet een A zijn.

Wordt aan slechts één van beide voorwaarden voldaan (of aan geen van beide), dan volgt de laatste printopdracht:

```
Volgende woord ..
```

Het spreekt voor zich dat er vele soorten voorwaarden mogelijk zijn. De procedure TEST1 is slechts een voorbeeld voor het gebruiken van AND.

● OR

```
EDIT "TEST2 :W
IF OR FIRST :W = "A FIRST :W = "a [PR [Dit woord begint
  met de eerste letter van het alfabet][Volgende woord ..]
END
```

Kijk maar:

```
TEST2 "andersom
Dit woord begint met de eerste letter van het alfabet

TEST2 "doorelkaar
Volgende woord ..
```

In dit voorbeeld zien we OR staan. Dit betekent dat er gekeken wordt of aan een van beide voorwaarden wordt voldaan. Is dit het geval dan wordt de eerste printopdracht uitgevoerd:

```
Dit woord begint met de eerste letter van het alfabet
```

De voorwaarden zijn:

- de eerste letter van het woord moet een A zijn;
- de eerste letter van het woord moet een a zijn.

Wordt aan geen van beide voorwaarden voldaan, dan verschijnt:

```
Volgende woord ..
```

Ook voor OR is een veelheid aan toepassingen te bedenken. De voorwaarden kunnen we steeds nader bepalen.

● NOT

```
EDIT "TEST3 :N
IF NOT NUMBERP :N [PR [Dit is geen getal !][PR [Dit
  is een getal]
END
```

Deze testopdracht draait de zaken om. Er staat eigenlijk:

Als :N GEEN getal is, voer dan de eerste opdracht uit.
Als :N WEL een getal is, voer dan de laatste opdracht uit.

Als we NUMBERP in de testopdracht een 'voorwaarde' noemen, dan

- geeft NOT de waarde TRUE als de voorwaarde FALSE is;
- geeft NOT de waarde FALSE als de voorwaarde TRUE is.

Kijk maar:

```
TEST3 40
Dit is een getal

TEST3 "Hallo
Dit is geen getal !
```

3.4 De Haiku.Generator

Genoeg theorie, we gaan nu eens kijken naar een heel aardig programma, dat het bovenstaande op uitstekende wijze illustreert. Eerst volgt de listing van het programma en daarna de uitleg.

En dan nu het programma: HAIKU.GENERATOR. Deze procedure maakt van de ingevoerde woorden steeds een ander gedicht: een *haiku*, ofwel een kort Japans gedichtje. Hier is de listing:

```
TO H.G :HOEVAAK?
IF :HOEVAAK? < "1 [STOP]
TS CT
MAKE "A [ochtend morgen dag avond nacht]
MAKE "B [gloort vervaagt verdwijnt ontwa
akt]
MAKE "C [nevel mist regen lichtkrans]
MAKE "D [veegt maakt vliegt lacht huilt
veronderstelt zwaait]
MAKE "E [bloem vlieg vlinder vlieger]
MAKE "F [lucht wolken hemel horizon]
MAKE "G [open schoon droog toe]
SETBG 56 + RANDOM 14
SETCURSOR [2 8] PR ( SE "De KIES :A KIES
 :B
SETCURSOR [2 10] PR ( SE "en "in "de KIE
S :C
SETCURSOR [2 12] PR ( SE KIES :D "een KI
ES :E "de KIES :F KIES :G
WAIT 200
H.G :HOEVAAK? - 1
END
```



```
TO KIES :LIJST
OP ITEM 1 + ( RANDOM COUNT :LIJST ) :LIJ
ST
END

TO ITEM :N :OBJECT
IF EMPTYP :OBJECT [OP "]
IF :N = 1 [OP FIRST :OBJECT]
OP ITEM :N - 1 BF :OBJECT
END
```

Dit zijn enkele voorbeeldjes van met de Haiku.Generator gemaakte haiku's:

```
De morgen verdwijnt
en in de lichtkrans
lacht een vlieger de lucht toe


De avond verdwijnt
en in de mist
maakt een bloem de hemel schoon


De morgen ontwaakt
en in de nevel
maakt een vlieger de lucht schoon


De morgen verdwijnt
en in de mist
veronderstelt een vlinder de hemel droog


De avond ontwaakt
en in de nevel
vliegt een vlieg de wolken open


De dag ontwaakt
en in de lichtkrans
maakt een vlieger de lucht schoon


De ochtend gloort
en in de mist
huilt een vlieger de hemel toe
```



De dag vervaagt
en in de lichtkrans
veronderstelt een vlieger de wolken droog

De nacht ontwaakt
en in de nevel
huilt een vlinder de wolken schoon

De avond gloort
en in de mist
veegt een bloem de horizon droog

De morgen verdwijnt
en in de lichtkrans
vliegt een vlieger de horizon open

De morgen ontwaakt
en in de regen
huilt een vlieg de hemel schoon

Uitleg van de Haiku Generator

Zoals u ziet bestaat het programma Haiku Generator uit drie procedures:
- de hoofdprocedure H.G (HAIKU.GENERATOR is zo'n lange naam);
- de subprocedure ITEM en
- de subprocedure KIES.

De procedures ITEM en KIES zijn op p.52 reeds besproken.

Om te beginnen vraagt deze procedure om een invoergegeven. De variabele :HOEVAAK? krijgt de waarde van dit invoergegeven. Doordat de waarde van :HOEVAAK? telkens, aan het eind van de procedure, met 1 wordt verminderd, fungeert :HOEVAAK? als een teller. H.G 10 betekent dus dat de procedure H.G tien maal wordt doorlopen.

H.G begint met een testopdracht, die de waarde van :HOEVAAK? bekijkt. Is deze waarde lager dan 1, dan stopt de procedure.

Vervolgens worden twee schermopdrachten gegeven: TS en CT. TS dient om aan te geven dat we gebruik maken van het Text Screen. Het gehele scherm wordt dan gebruikt voor tekst. CT dient om het scherm schoon te maken met Clear Text.

Hierna wordt aan de variabelen :A, :B, :C, :D, :E, :F en :G een waarde toegekend. Deze waarde bestaat uit een lijst met vier, vijf of zeven elementen. Indien u wilt experimenteren kunt u de elementen in deze lijsten rustig aanpassen of veranderen.

SETBG 56 + RANDOM 14 is wederom een opdracht voor het beeldscherm. Hiermee bepalen we de kleur van het beeldscherm op RANDOM-wijze.

Nu volgen de eigenlijke hoofdregels van de procedure. Na de SETCURSOR-opdracht volgt:

```
PR (SE "De KIES :A KIES :B
```

Deze PRINT-opdracht heeft tot gevolg dat
- eerst het woord De wordt afgedrukt;
  (Opmerking. Vergeet niet om met " aan te geven dat het om een woord gaat.)
- daarna de procedure KIES wordt aangeroepen met als invoer :A. Omdat :A de waarde [ochtend morgen dag avond nacht] heeft, zal een van deze woorden worden gekozen.
- tenslotte op dezelfde manier uit de variabele :B een woord wordt gekozen.

Vervolgens komen de variabelen :C, :D, :E, :F en :G aan de beurt. De, van tevoren bepaalde, voorzetsels, lid- en voegwoorden worden aan de RANDOM-woorden 'geplakt' met de instructie SE. Let er wel op dat aan SE een (moet voorafgaan.

De procedure besluit met een wacht-opdracht van 200 eenheden: WAIT 200. Als 'eenheid' wordt in LCSI Logo 1/60 seconde gebruikt.

Tenslotte wordt de, recursieve, procedure opnieuw aangeroepen en wordt de waarde van :HOEVAAK? met 1 verminderd.

3.5 Invoeropdrachten met Read Character en Read List

In veel taalprocedures zult u de opdrachten RC en RL tegenkomen. Dit is ook het geval in de procedures TEGENSTELLING, PSYCHIATER en de databases LOGOMEMO en LOGOMANAGER. Maar voordat we naar deze procedures gaan kijken, richten we onze aandacht eerst op de opdrachten

RC = Read Character
RL = Read List.

Deze opdrachten werden reeds invoeropdrachten genoemd. Dit betekent dat beide pas geactiveerd worden wanneer er iets wordt ingevoerd. Deze invoer is
- bij RC een teken
- bij RL een lijst (met een of meer elementen).

Eerst gaan we eens naar de opdracht RC kijken.

De opdracht RC (Read Character)

Ter illustratie dit voorbeeld:

```
EDIT "SOMMEN
MAKE "GETAL1 RANDOM 5
MAKE "GETAL2 RANDOM 5
MAKE "SOM :GETAL1 + :GETAL2
PR [Hier komt een som :]
PR []
TYPE (SE :GETAL1 "+ :GETAL2 "= "
MAKE "OPLOSSING RC
TYPE :OPLOSSING
PR []
IF EQUALP :OPLOSSING :SOM [PR [Okee !]][PR [Fout]]
PR [Wil je doorgaan J / N ?]
MAKE "KEUS RC
IF :KEUS = "J [PR [Okee dan gaan we verder] SOMMEN][PR
[Dan stoppen we]
END
```

In deze procedure wordt eerst de uitkomst van een som gevraagd. Met de opdracht RC wordt het antwoord toegekend aan de variabele OPLOSSING:

```
MAKE "OPLOSSING RC
```

Hier 'wacht' de procedure tot er iets wordt ingevoerd.

Nadat het teken is ingevoerd, wordt het vergeleken met de waarde van de variabele :SOM. En als gekeken is of het gegeven antwoord klopte, wordt de vraag gesteld:

```
Wil je doorgaan J / N ?
```

Ook hier wacht de procedure op invoer. Dat komt door:

```
MAKE "KEUS RC
```

Er wordt dus een variabele :KEUS gemaakt die de waarde krijgt van de toets die wordt ingedrukt. Wordt de 'J' ingedrukt, dan zorgt RC ervoor, dat de variabele :KEUS de waarde J krijgt. Wordt er echter op bijvoorbeeld de * gedrukt, dan zal :KEUS de waarde * ontvangen. En als :KEUS een andere waarde dan J krijgt, zal de tweede printopdracht worden uitgevoerd.

Op het moment dat aan de variabele :KEUS een waarde is toegekend, dus op het moment dat er een toets wordt ingedrukt, zal de procedure verder gaan met de IF-opdracht.

Is de waarde van :KEUS gelijk aan "J (denk om de aanhalingstekens, anders denkt Logo dat J de naam van een procedure is), dan zal de eerste opdracht worden uitgevoerd. Deze opdracht drukt de zin

```
Okee dan gaan we verder
```

af. Is echter de waarde van :KEUS ongelijk aan J (het maakt geen verschil of dat N is of een andere toets), dan zal de tweede opdracht worden uitgevoerd.

Opmerking. De tweede printopdracht wordt dus uitgevoerd bij iedere waarde van :KEUS, behalve bij J. In plaats van de N mag dus gerust een andere toets worden ingedrukt.

Nog een voorbeeld:

```
PR [Als je wilt stoppen Toets RETURN]
MAKE "KEUS RC
IF :KEUS = CHAR 155 [STOP] [PR [We gaan verder]
```

Een teken kan dus ook aangeduid worden met zijn ASCII-codenummer, in dit geval 155.

De opdracht RL (Read List)

Nu kijken we eens naar het volgende programmafragment:

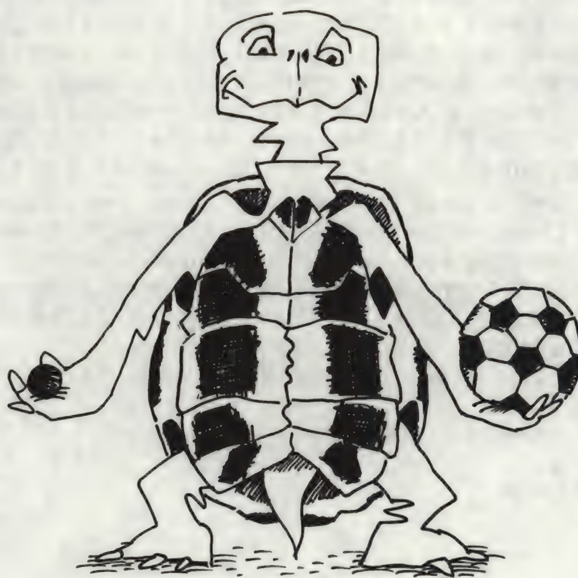
```
PR [Hallo , hoe is je naam ?]
MAKE "NAAM RL
PR ( SE [Leuk dat je hier bent] :NAAM
PR [Sorry dat ik zo nieuwsgierig ben,]
PR [maar , waar woon je ?]
MAKE "WOONPLAATS RL
IF :WOONPLAATS = [Dokkum] [PR [Daar kom ik vaak]]
PR ( SE [Dus] :NAAM "woont "in :WOONPLAATS
```

U ziet het, RL kent aan de variabele :NAAM de waarde van een in te toetsen naam toe en :WOONPLAATS krijgt de waarde van een volgende ingetoetste naam. Het resultaat van het bovenstaande kan dus zijn:

```
Hallo , hoe is je naam ?
Auke Sikma
Leuk dat je hier bent Auke Sikma
Sorry dat ik zo nieuwsgierig ben,
maar , waar woon je ?
Rinsumageest
Dus Auke Sikma woont in Rinsumageest
```

Zo kunnen zowel RC als RL zeer zinvol zijn in procedures.

3.6 De tegenstelling



Een voorbeeld van een zinvol gebruik van de opdrachten RC en RL vindt u in de procedure TEGENSTELLING. Deze procedure, die beschikt over een uitgebreide woordenschat, kan een zin, een verhaal zo u wilt, volledig herschrijven. Maar dan in het TEGENOVERGESTLEDE! Alles wat dik was wordt dun, alles wat arm was wordt rijk, enzovoort.

Om een zin te kunnen herschrijven zijn de opdrachten RL, FIRST, LAST, BF en FPUT heel belangrijk.

Bekijkt u een en ander maar eens rustig.

```
TO VERGELIJK :regel :woorden
IF EMPTYP :woorden [OP :regel]
IF :regel = FIRST FIRST :woorden [OP LAST FIRST :w
oorden] [OP VERGELIJK :regel BF :woorden]
END


TO N.ZIN :regel :woorden
IF EMPTYP :regel [OP []]
OP FPUT ( VERGELIJK FIRST :regel :woorden ) ( N.ZI
N ( BF :regel ) :woorden )
END


TO TEGENSTELLING
TS CT SETBG 0
MAKE "woorden [[slimme domme] [domme slimme] [oude
 nieuwe] [nieuwe oude] [jonge oude] [oude jonge] [
blije verdrietige] [verdrietige blije] [tamme wild
e] [wilde tamme] [grote kleine] [kleine grote] [du
re goedkope] [goedkope dure] [domme intelligente]
[intelligente domme] [hoge lage] [lage hoge] [veel
 weinig] [weinig veel] [arme rijke] [rijke arme] [
atari commodore] [commodore atari] [kat hond] [hon
d kat] [schone vieze] [vieze schone] [volle lege]
[lege volle] [gewone andere] [andere gewone] [lang
e korte] [korte lange] [koffie thee] [thee koffie]
 [knappe lelijke] [lelijke knappe] [dikke dunne] [
dunne dikke] [goede slechte] [slechte goede] [vrie
ndelijke boze] [boze vriendelijke] [lichte zware]
[zware lichte]]
PR [Welke zin moet ik herschrijven ?]
MAKE "regel RL
PR N.ZIN :regel :woorden
WAIT 500
TEGENSTELLING
END
```

Zoals u in de subprocedure VERGELIJK kunt zien, wordt een zin woord voor woord bekeken.

1. Er wordt gekeken of het eerste woord van :REGEL (de ingetoetste zin) overeenkomt met het eerste element van het eerste element in :WOORD. Dit gebeurt door:
   IF :REGEL = FIRST FIRST :WOORD
   Als dit het geval is, wordt dit woord vervangen door het laatste element van het eerste element, als volgt:
   OP LAST FIRST :WOORD
   Met andere woorden: als de ingetoetste zin begint met Slimme, wordt dit vervangen door Domme.
2. Komt het eerste woord van :REGEL niet overeen met het eerste van het eerste element van :WOORD, dan wordt het eerste element van :WOORD terzijde gelegd met
   OP VERGELIJK :REGEL BF :WOORD
   Hierna wordt het eerste woord van :REGEL vergeleken met het eerste element van het tweede element van :WOORD. Enzovoort.
3. Zo wordt ieder woord van de ingetoetste zin vergeleken met het eerste element van alle elementen van :WOORD en zo mogelijk vervangen door het tweede element van dat element.
4. Uiteindelijk wordt de gehele zin als het ware herschreven. Bijvoorbeeld:

- De grote intelligente schrijver kocht een dure maar prima printer.
- De kleine domme schrijver kocht een goedkope maar slechte printer.

Of:

- De dikke maar tamme kater was verliefd op de korte knappe poes.
- De dunne maar wilde kater was verliefd op de lange lelijke poes.

Op dezelfde manier is natuurlijk ook een procedure SYNONIEMEN te maken. Dan zou een woord niet worden vervangen door het tegenovergestelde maar door een ander woord met ongeveer dezelfde betekenis. Een lijst zou er dan zo uit kunnen zien:

```
MAKE "SYNONIEMEN [[verdrietige droevige][kleine
minuscule][grote reusachtige], enzovoort .....
```

In wezen verandert de procedure niet, alleen zou de variabele :WOORD vervangen moeten worden door een andere variabele. Of de variabele :WOORD zou een andere waarde moeten krijgen.

3.7 De Logo-psychiater



Een fantastisch programma, vinden velen, is PSYCHIATER. Dit programma simuleert een gesprek tussen u (als patiënt) en een psychiater. Het programma is niet alleen leuk om te doen, maar ook om te onderzoeken, om mee te experimenteren, als u dat wilt. Woorden, zinnen, antwoorden en rake opmerkingen toevoegen, u ziet maar.

Hier is onze Logo Psychiater.

```
TO PSYCHIATER
TS CT SETBG 0
MAKE "COMMENTAAR [[Je denkt ,] [Je voelt,] [Je zeg
t ,] [Ik denk ,] [Ik weet ,] [Ik veronderstel ,] [
Je vertelt me] [Je zegt me] [Je wilt me doen gelov
en] [Zonder blikken of blozen vertel je mij] [Je w
ilt zeggen] [Hoor ik dat goed ...] [Denk je echt .
.]]
MAKE "ZIN [[Ja,ja,ga verder...] [Niet bang zijn,ze
g het maar..] [Daar hebben veel mensen last van] [
Zo,zo..] [Kijk eens aan...] [Je meent het...] [Is
dat echt waar ?] [Ongelovelijk...] [Ik heb veel me
egemaakt , maar dit ...] [Dat kun jij makkelijk ze
ggen...] [Daar hebben we het nu niet over..] [Dat
heeft er niets mee te maken ..] [Ja luister eens ,
als je zo begint ..] [Daar zou je toch eens wat aa
n moeten doen] [En wat kan ik daaraan doen ?] [Vin
d je dat leuk om te vertellen] [Daar schiet ik nie
ts mee op natuurlijk] [Wat is dat nou voor een onz
in ?] [Niet er omheen draaien] [De waarheid graag
...] [Ik heb een iemand gekend die had dat ook]]
MAKE "DRAAIOM [[uw mijn] [vraag vraagt] [kunt kan]
 [ik je] [mij jou] [jou mij] [u ik] [ik u] [geef g
eeft] [mijn jouw] [ben bent] [me je] [je ik] [voel
 voelt] [jouw mijn] [kan kunt] [denk denkt] [heb h
ebt] [zie ziet] [slaap slaapt] [locp loopt] [fiets
 fietst] [droom droomt] [geloof gelooft] [veronder
stel veronderstelt] [beweer beweert] [hoor hoort]]
PR [Daag ,hoe heet je ?]
MAKE "NAAM RL
PR SE [Wat scheelt eraan] :NAAM
COMMUNICEREN
END

TO COMMUNICEREN
MAKE "INVOER RL
IF :INVOER = [tot ziens] [PR [Kom nog maar eens la
ngs] STOP]
MAKE "X .EXAMINE 53770
IF :X > 100 [PR KIES :ZIN]
IF :X < 101 [ANTWOORD :INVOER]
COMMUNICEREN
END

TO KIES :LIJST
OP ITEM 1 + ( RANDOM COUNT :LIJST ) :LIJST
END
```

```
TO ANTWOORD :INVOER
PR ( SE ( KIES :COMMENTAAR ) ( VERANDER :INVOER :D
RAAIOM
END

TO ITEM :N :OBJECT
IF EMPTYP :OBJECT [OP "]
IF :N = 1 [OP FIRST :OBJECT]
OP ITEM :N - 1 BF :OBJECT
END

TO VERGELIJK :INVOER :DRAAIOM
IF EMPTYP :DRAAIOM [OP :INVOER]
IF :INVOER = FIRST FIRST :DRAAIOM [OP LAST FIRST :
DRAAIOM] [OP VERGELIJK :INVOER BF :DRAAIOM]
END

TO VERANDER :INVOER :DRAAIOM
IF EMPTYP :INVOER [OP []]
OP FPUT ( VERGELIJK FIRST :INVOER :DRAAIOM ) ( VER
ANDER ( BF :INVOER ) :DRAAIOM )
END
```

Uitleg van het programma PSYCHIATER

Wat gebeurt er wanneer we de procedure PSYCHIATER starten?

1. Na enige schermopdrachten (TS CT en het bepalen van de beeldschermkleur) worden de variabelen 'gevuld'. De variabelen :COMMENTAAR, :ZIN en :DRAAIOM krijgen hier hun waarde. Elk van deze variabelen bestaat nu uit een lijst met meerdere elementen.
2. Hierna wordt de openingszin op het scherm gezet: Daag, hoe heet je ? Het antwoord op deze vraag wordt aan de variabele :NAAM toegekend, en het programma vervolgt met de zin: Wat scheelt eraan ... (gevolgd door de inhoud van de variabele :NAAM).
3. Op het moment dat de procedure COMMUNICEREN wordt aangeroepen begint het eigenlijke deel van het programma, het hoofdprogramma.
4. COMMUNICEREN is een recursieve procedure die steeds, door RL, de ingetypte zin als uitgangspunt neemt. Deze ingetypte zin is in het programma de variabele :INVOER. Wanneer "tot ziens" wordt ingetypt, stopt de procedure en zegt de Logo-Psychiater: Kom nog maar eens langs.
5. Vervolgens wordt, binnen COMMUNICEREN, op RANDOM-wijze de waarde van :X bepaald. Dit gebeurt hier niet met de opdracht RANDOM, maar met het uitlezen (opdracht .EXAMINE) van adres 53770. Adres 53770 is (voor Atari 8-bits computers) het adres waar zich de toevalsgetallen bevinden. Indien u met een andere Logo-versie werkt, gebruik dan het juiste adres voor uw computer.

Door MAKE "X .EXAMINE 53700 wordt aan de variabele :X een waarde toegekend tussen 0 en 255. Is dit cijfer lager dan 101 dan treedt de procedure ANTWOORD in werking en wordt de ingetypte zin veranderd en aangepast. Is de waarde van :X groter dan 100, dan kiest de Psychiater een zin. Dit zijn steeds zinnen die niets te maken hebben met wat er gezegd is, maar juist daardoor kan dit wel een komisch effect hebben. Kijk maar eens naar het voorbeeld van een PSYCHIATER-dialoog op deze en de volgende bladzijde.

6. Wanneer de voor :X bepaalde waarde kleiner is dan 101 treden de procedures ANTWOORD, VERGELIJK en VERANDER in werking.
7. De procedure ANTWOORD, met de ingetypte zin als uitgangspunt, zorgt via KIES :COMMENTAAR eerst voor een aardige opening van de zin. Hierna wordt door VERANDER en VERGELIJK de hele zin doorgelicht en op dezelfde manier behandeld als in de procedures bij TEGENSTELLING.

   Treft VERGELIJK in de ingetypte zin een woord aan dat overeenkomt met het eerste element van een der elementen van :DRAAIOM, dan wordt dat woord veranderd in het tweede element van dat element. Met andere woorden: treft VERGELIJK het woord "kunt aan in de ingetypte zin, dan wordt dit ogenblikkelijk veranderd in "kan; je verandert in ik, mij in jou, enzovoort. Op deze manier wordt de hele zin bekeken en waar nodig veranderd.

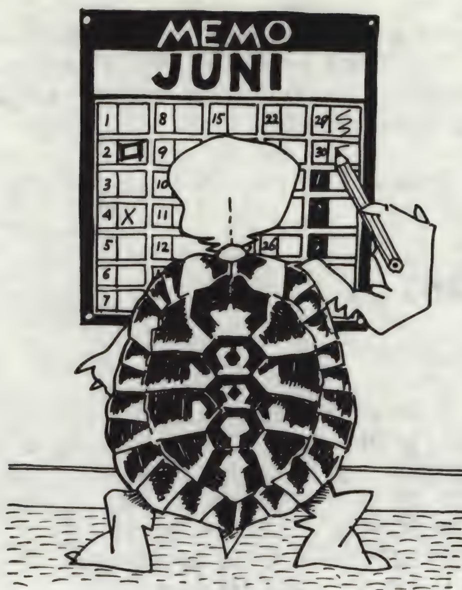
   Kunt u zich voorstellen dat, wanneer de verzameling woorden maar groot genoeg is, er een goed gesprek mee te voeren is?
8. Tenslotte wordt de procedure COMMUNICEREN opnieuw geactiveerd. Dit gaat net zo lang door totdat u "tot ziens" intypt.

Hierna volgt een voorbeeld van een dialoog met de Logo-Psychiater. Veel plezier ermee!

```
?PSYCHIATER
Daag ,hoe heet je ?
Auke Sikma
Wat scheelt eraan Auke Sikma
ik heb hoofdpijn
Je voelt, je hebt hoofdpijn
ik denk dat ik het te druk heb
Wat is dat nou voor een onzin ?
ik ben namelijk een boek aan het schrijven
Is dat echt waar ?
ja , een boek over Logo
Niet bang zijn,zeg het maar..
ik denk dat ik me te druk maak
Daar hebben we het nu niet over..
nee , maar dat is wel mijn probleem
Dat heeft er niets mee te maken ..
dat heeft er verdorie alles mee te maken      →
```

Ja luister eens ,als je zo begint ..
kunt u mij helpen
Niet er omheen draaien
ik vraag u iets
Je zegt me je vraagt ik iets
wat praat u raar dokter
Dat heeft er niets mee te maken ..
dat is waar
Ik denk , dat is waar
er is nog iets ....
De waarheid graag ...
ik droom nogal vreemd de laatste tijd
Je denkt , je droomt nogal vreemd de laatste tijd
ik droom over slechte recensies
Ja luister eens ,als je zo begint ..
daar ben ik bang voor dokter
Je zegt me daar bent je bang voor dokter
nu heeft u dat weer , dat vreemde gepraat
Vind je dat leuk om te vertellen
nou nee , maar het valt wel op
Dat kun jij makkelijk zeggen...
ja , want ik heb u gemaakt
Dat heeft er niets mee te maken ..
o , nee ??
Je meent het...
ik voel me onzeker
Daar schiet ik niets mee op natuurlijk
kunt u mij helpen ?
Niet er omheen draaien
ik wil graag rustig slapen
Je zegt me je wil graag rustig slapen
ja , en niet meer zo raar dromen
Ik denk , ja , en niet meer zo raar dromen
dat is wat ik wil
Je wilt zeggen dat is wat je wil
ik zie dat u mij niet helpt
Je vertelt me je ziet dat ik jou niet helpt
uw Nederlands is niet om over naar huis te schrijven
Hoor ik dat goed ... mijn Nederlands is niet om over naar huis te schrijven
ja , dat zei ik
En wat kan ik daaraan doen ?
niets , dat is mijn schuld
En wat kan ik daaraan doen ?
ik denk dat ik maar eens ga
Hoor ik dat goed ... je denkt dat je maar eens ga
o,o,o,o,o
Ja luister eens ,als je zo begint ..
tot ziens
Kom nog maar eens langs

3.8 Logo en databases



Logo is wellicht niet de meest logische keus voor het maken van een database (een elektronische kaartenbak), maar toch kan het heel erg leerzaam zijn hier even bij stil te staan.

Allereerst het programma "LOGOMEMO. LOGOMEMO is een zeer eenvoudige programma met verschillende opties:

1. Het schrijven van een memo.
2. Het lezen van een memo.
3. Een overzicht van alle memo's.
4. Het afdrukken van een memo.
5. Het wissen van een memo.
6. Het laden van een memo.
7. Het opslaan van een memo.

LOGOMEMO is gebaseerd op het gebruik van een disk-drive. Databases die met bandjes werken zijn te traag en daardoor onhandig in het gebruik.

LOGOMEMO is een database-programma dat ieder bestand (ieder memo, ieder gegeven) apart wegschrijft. Later zullen we zien dat memo's ook per groep weggeschreven kunnen worden, bijvoorbeeld door de variabele :MEMO'S te maken. Ieder nieuw memo wordt hier door middel van de opdracht LPUT in opgenomen. Op deze manier

kunt u met LOGOMEMO experimenteren en er uiteindelijk uw eigen database mee opzetten. In verband hiermee zullen we later in dit boek dan ook sorteerroutines gaan bekijken, want elke goede database moet zijn gegevens kunnen sorteren.

LOGOMEMO is tevens interessant om te kijken over welke Disk- en Printer-opdrachten Logo beschikt. SETREAD en SETWRITE bijvoorbeeld zijn prachtige Logo-opdrachten, vooral wanneer ze met een printer worden gehanteerd.

Laten we nu eens gaan kijken naar het eigenlijke programma.

3.9 LOGOMEMO

```
TO LOGOMEMO
CT TS SETBG 0 ERN [KEUS VERDER]
SETCURSOR [5 1] PR [. . . L o g o m e m o . . .]
SETCURSOR [11 5] PR [1.Schrijven]
SETCURSOR [11 7] PR [2.Lezen]
SETCURSOR [11 9] PR [3.Overzicht]
SETCURSOR [11 11] PR [4.Afdrukken]
SETCURSOR [11 13] PR [5.Wissen]
SETCURSOR [11 15] PR [6.Laden]
SETCURSOR [11 17] PR [7.Opslaan]
SETCURSOR [5 20] PR [Doe je keus maar ......]
MAKE "KEUS RC
IF :KEUS = "1 [SCHRIJF]
IF :KEUS = "2 [LEES]
IF :KEUS = "3 [ZIE]
IF :KEUS = "4 [PRNT]
IF :KEUS = "5 [WIS]
IF :KEUS = "6 [LAAD]
IF :KEUS = "7 [WSCHRIJF]
END


TO SCHRIJF
CT TS
PR [Schrijven :]
PR []
PR []
PR [* Schrijf hier je memo :]
PR []
PR []
MAKE "MEMO RL
PR [Toets RETURN in om door te gaan]
MAKE "VERDER RC
IF :VERDER = CHAR 155 [CT ERN :VERDER LOGOMEMO]
END
```



```
TO LEES
CT TS
PR [Lezen :]
PR []
PR []
IF NOT NAMEP "MEMO [PR [Er zit geen memo in het ge
heugen] WAIT 200 LOGOMEMO]
PR :MEMO
MAKE "VERDER RC
IF :VERDER = CHAR 155 [CT ERN :VERDER LOGOMEMO]
END


TO ZIE
CT TS
PR [Overzicht :]
PR []
PR []
PR [Deze memo's staan op de disk :]
CATALOG "D:
PR [Toets RETURN om door te gaan]
MAKE "VERDER RC
IF :VERDER = CHAR 155 [CT ERN :VERDER LOGOMEMO]
END


TO PRNT
CT TS
IF NOT NAMEP "MEMO [PR [Er zit geen memo in het ge
heugen] WAIT 200 LOGOMEMO]
PR [Afdrukken :]
PR []
PR [Druk RETURN om deze memo af te drukken :]
PR []
PR []
PR []
MAKE "VERDER RC
IF :VERDER = CHAR 155 [ERN :VERDER SETWRITE "P:]
.DEPOSIT 559 0
PR :MEMO
SETWRITE []
.DEPOSIT 559 58
PR []
PR []
PR [Terug naar menu met RETURN]
MAKE "VERDER RC
IF :VERDER = CHAR 155 [CT ERN :VERDER LOGOMEMO]
END
```

→

```
TO WIS
CT TS
IF NOT NAMEP "MEMO [PR [Er zit geen memo in het ge
heugen] WAIT 200 LOGOMEMO]
PR [Wissen :]
PR []
PR [Moet ik deze memo wissen J / N ?]
PR []
PR []
PR :MEMO
MAKE "KEUS RC
IF :KEUS = "J [ERNS LOGOMEMO] [LOGOMEMO]
END


TO LAAD
CT TS
PR [Geef de naam van de te laden memo :]
MAKE "NAAM RL
PR []
PR ( SE [De memo :] :NAAM
PR [wordt nu geladen .....]
SETREAD WORD "D: FIRST :NAAM
MAKE "MEMO RL
REPEAT 3 [PR RL]
SETREAD []
LOGOMEMO
END


TO WSCHRIJF
CT TS
PR [Opslaan :]
PR []
PR []
IF NOT NAMEP "MEMO [PR [Er zit geen memo in het ge
heugen] WAIT 200 LOGOMEMO]
PR [Deze memo opslaan ?]
PR []
PR []
PR :MEMO
PR []
PR [Onder welke naam ?]
MAKE "NAAM RL
PR []
PR ( SE [Memo] :NAAM
PR [wordt nu weggeschreven.]
SETWRITE WORD "D: FIRST :NAAM
WAIT 50 .DEPOSIT 559 0
PR :MEMO
```

→

```
SETWRITE []
CT .DEPOSIT 559 58
LOGOMEMO
END
```

LOGOMEMO, woord voor woord

1. De procedure LOGOMEMO is de hoofdprocedure; deze laat (na enkele opdrachten voor het beeldscherm en het wissen van de variabelen :KEUS en :VERDER met ERN = ERase Name) op het scherm een keurig keuzemenu zien. Door het intoetsen van een getal kan een keuze worden gemaakt. Wanneer deze keuze is gemaakt wordt met de opdracht RC en de variabele :KEUS een subprocedure gekozen.
2. De procedure SCHRIJF is heel eenvoudig. Hierin wordt de variabele :MEMO gemaakt die door de opdracht RL de waarde krijgt van de ingetoetste boodschap. Is de memo ingetypt dan gaat het programma (zie in de procedure de opdracht RC en CHAR 155 (de ASCII-code van de RETURN-toets)) terug naar het menu. Meteen wordt de variabele :VERDER ook weer gewist met ERN.
3. De procedure LEES begint met een testopdracht waarin we NAMEP weer tegenkomen. Deze testopdracht kijkt namelijk of de variabele :MEMO wel een waarde heeft (dus of :MEMO wel bestaat). Als we proberen een memo te lezen zonder dat we er een geschreven (of geladen) hebben, verschijnt de boodschap: "Er zit geen memo in het geheugen" op het scherm of op de printer. Bestaat er wel een variabele :MEMO, dan wordt de waarde hiervan (de ingetypte boodschap dus) keurig afgedrukt.
   De procedure gaat tenslotte, net als alle andere, terug naar het menu.
4. De procedure ZIE laat door middel van de opdracht CATALOG "D: de diskette-inhoud zien. Naast de weggeschreven memo's zien we dan ook de LOGOMEMO-procedures staan (tenzij we voor de memo's een aparte diskette gebruiken). De opdracht CATALOG is alleen te gebruiken met een (of meer) disk-drive(s). Bij een tweede drive moet CATALOG "D2: worden ingetoetst.
5. Het zou handig geweest zijn als we de procedure PRNT de naam PRINT hadden kunnen geven, maar dat kan niet, want PRINT is al een Logo-woord. Ook PRNT kijkt eerst of er wel een memo bestaat, en legt de verbinding met de printer met SETWRITE "P: . Deze opdracht zorgt ervoor dat alles wat op het scherm verschijnt afgedrukt wordt op de printer. Even verderop zien we de opdracht PR :MEMO staan. Deze heeft tot gevolg dat :MEMO inderdaad verschijnt, zowel op het beeldscherm als op papier.
6. De procedure WIS begint met de, inmiddels bekende, testopdracht. Immers als er geen memo is kan er ook geen memo gewist worden. Het wissen gebeurt daarna met ERNS = ERase NameS. We zouden de procedure WIS ook zo kunnen veranderen dat er

een memo uit het geheugen (met ERN) of een memo van de diskette gewist wordt. Een memo-bestand van diskette wissen we met ERF "D:bestandsnaam.

7. De procedure LAAD maakt gebruik van de opdracht SETREAD. De opdracht LOAD zou hier namelijk niet de juiste opdracht zijn. Door WORD (na SETREAD) wordt de bestandsnaam gekoppeld aan de apparaatnaam (D:) (D staat voor Device). Nadat de memo op het scherm verschenen is wordt met MAKE "MEMO RL de inhoud van de memo als het ware aan de variabele :MEMO gekoppeld. Daarna maakt SETREAD [ ] de 'leesverbinding' met de drive.
8. De procedure WSCHRIJF (die we ook geen SAVE kunnen noemen!) begint met de bekende testopdracht: IF NOT NAMEP "MEMO ... Daarna krijgt de variabele :NAAM de naam van de memo en wordt met behulp van SETWRITE "D: opgeslagen ('gesaved'). Ook hier worden weer de Poke-waarden uit Logo gebruikt om het beeldscherm even uit te schakelen en zo het opslagproces te versnellen. Met .DEPOSIT 559 58 wordt alles weer normaal en met SETWRITE [ ] wordt de verbinding met de disk-drive verbroken. Vergeet dit niet!!

3.10 Logo-Manager; stap voor stap uw eigen database maken

In deze paragraaf gaan we bekijken hoe we zelf een gegevensbestand in Logo kunnen maken. U krijgt de bouwstenen waarmee u uw eigen database kunt opzetten. Zo krijgt u een Logo-Manager naar uw eigen wensen en verlangens. U krijgt dus geen uitgewerkte database, maar wel uitgewerkte zoek-, sorteer-, laad-, opslag-, print- en telroutines.

Indien u het voorafgaande, vooral de Logomemo, goed hebt doorgenomen, is het een kleine moeite uw eigen database samen te stellen. De namen van de procedures en variabelen zijn willekeurig gekozen. Het staat u dus geheel vrij er uw eigen namen aan te geven.

Logo-Manager is een database met een andere opzet dan Logomemo. Bij Logomemo werden de gegevens steeds apart opgeslagen; bij Logo-Manager worden de gegevens in een lijst geplaatst en deze lijst wordt in zijn geheel weggeschreven.

Om alle gegevens in een lijst te plaatsen maken we gebruik van de opdracht LPUT. Kijkt u eens naar deze procedures:

```
EDIT "setup              EDIT "feit :1
MAKE "feiten []          MAKE "feiten LPUT :1 :feiten
END                      END
```

```
EDIT "feiten
IF EMPTYP :feiten [PR [...geen feiten]]
 [printfeiten :feiten]
END

EDIT "printfeiten :l
IF EMPTYP :l [STOP]
TYPE [*] PR FIRST :l
printfeiten BF :l
END
```

Deze vier eenvoudige routines vormen de basis van de gehele database. Door de procedure "feit wordt ieder nieuw stukje informatie aan het einde van een lijst geplaatst. Kijkt u eens naar het gevolg van deze procedures:

```
?setup
?feit [Jan Vis Dorpstraat 8 Laren]
?feit [Henk Kok Kerkstraat 2 Aalst]
?feit [Lia Jongman 't Pad 27 Breda]
?feiten
*Jan Vis Dorpstraat 8 Laren
*Henk Kok Kerkstraat 2 Aalst
*Lia Jongman 't Pad 27 Breda
?setup
?feiten
...geen feiten
```

Dit ziet er heel eenvoudig uit, maar toch is dit het grondbeginsel van deze database: een nieuw gegeven wordt aan het einde van een lijst geplaatst.

Natuurlijk moeten er voor een echte database ook zoek- en sorteerroutines komen, maar dit is pas het begin.

Laten we nu eens gaan kijken naar een zoekroutine. We noemen deze zoekroutine gewoon "zoek, het te zoeken gegeven :z en de lijst waaruit we moeten zoeken :feiten. De procedure lijkt heel sterk op de zoekprocedures die we al bij TEGENSTELLING en PSYCHIATER zijn tegengekomen.

Om het maar eens eenvoudig te zeggen: "zoek neemt het gegeven :z als uitgangspunt en bekijkt alle elementen van de lijst :feiten. Is er een element waarvan het eerste gedeelte overeenkomt met het gegeven :z dan wordt dit element afgedrukt. Dit zoeken gaat razendsnel, zelfs met een grote lijst.

Een beperking van "zoek is, dat de procedure alleen kan zoeken met het eerste gegeven van een element als uitgangspunt. Dit kunt u echter in een handomdraai veranderen. Door naast FIRST ook FIRST BF te gebruiken kunt u elk willekeurig gegeven uit de lijst als uitgangspunt nemen voor een zoekroutine. Voor het gemak is hier voor de volgende vorm gekozen:

```
EDIT "zoek :z :feiten
IF EMPTYP :feiten [STOP]
IF :z = FIRST FIRST :feiten [PR FIRST :feiten zoek
 :z BF :feiten] [zoek :z
BF :feiten]
END
```

En dat is het hele verhaal. Deze procedure 'zoekt' in de gehele lijst :feiten of het gegeven :z erin staat. Kijk maar:

```
?zoek "Jan :feiten
Jan Vis Dorpstraat 8 Laren
?feit [Jan de Groot Laan 33 Holst]
?zoek "Jan :feiten
Jan Vis Dorpstraat 8 Laren
Jan de Groot Laan 33 Holst
?
```

Willen we even kijken hoeveel gegevens er beschikbaar zijn? Dat is geen enkel probleem, we tellen gewoon de elementen van de lijst. Kijk maar:

```
EDIT "tel :n
PR (SE [Er zijn momenteel :] COUNT :n "gegevens
 "beschikbaar
END
```

De opdracht COUNT telt gewoon alle elementen van de gegeven lijst.

Iets ingewikkelder wordt het wanneer we een sorteerroutine gaan bekijken. Een sorteerroutine is een procedure die de gegevens van de lijst :feiten alfabetisch rangschikt.

De hoofdprocedure zou er zo uit kunnen zien:

```
TO sorteer :n
MAKE "gesorteerd sort :n []
PR [Gegevens tonen J / N]
MAKE "toon RC
IF OR :toon = "J :toon = "j [PR ssort :gesorteerd]
 [STOP]
END
```

U ziet het, de door ..sort.. gesorteerde lijst wordt "gesorteerd genoemd en op verzoek door ssort op het scherm getoond.

De in de sorteerroutine voorkomende subprocedures sort, in, voor en ssort zien er als volgt uit:

```
TO in :a :b
IF EMPTYP :b [OP FPUT [] LIST :a []]
IF voor :a FIRST BF :b [OP FPUT in :a FIRST :b BF
:b]
OP LPUT in :a LAST :b BL :b
END

TO ssort :a
IF EQUALP :a [] [OP []]
OP ( SE ssort FIRST :a FIRST BF :a ssort LAST :a
END

EDIT "voor :a :b
IF OR EMPTYP :a EMPTYP :b [OP EMPTYP :a]
IF NOT EQUALP FIRST :a FIRST :b [OP (ASCII FIRST :a)
 < (ASCII FIRST :b)]
OP voor BF :a BF :b
END

TO sort :a :b
IF EMPTYP :a [OP :b]
MAKE "b in FIRST :a :b
OP sort BF :a :b
END
```

U merkt wel dat de procedures zo langzamerhand iets minder eenvoudig worden. Het is echter een kwestie van ervaring. Als u maar regelmatig oefent, leest u na een tijdje ook de regel:

```
IF NOT EQUALP FIRST :a FIRST :b [OP (ASCII FIRST :a)
 < (ASCII FIRST :b)]
```

met gemak als:

> indien het eerste teken (of element) van :a ongelijk is aan dat van :b, geeft dit het eerste teken (of element) van :a een lagere ASCII-code dan dat van :b.

En daarmee hebben we meteen de essentie van de sorteerroutine te pakken: alfabetisch rangschikken op grond van de ASCII-waarde van de tekens.

Voor de printroutines, de laad- en opslagprocedures kunt u de desbetreffende opdrachten uit Logomemo gebruiken. U moet deze dan wel aanpassen! Denkt u hierbij de opdrachten SETWRITE en SETREAD te gebruiken, denk dan om de 'sluitopdracht' SETREAD [ ] of SETWRITE [ ].

Met behulp van de gegeven procedures kunt u beslist een Logo-Manager maken die volledig is aangepast aan uw eigen wensen en

verlangens. Bovendien doorziet u een gegevensbestand veel beter wanneer u het zelf hebt opgezet.

Om u een indruk te geven van de hoofdprocedure volgt hier een voorbeeld van een mogelijke opzet. Het advies is echter: schrijf/bewerk eerst de subprocedures en controleer deze op hun werking. Schrijf de hoofdprocedure pas als alle subprocedures klaar zijn (de sorteer-, de print-, de wis-, de tel-, de opslagprocedure, enzovoort).

Logo-Manager, voorbeeld van een opzet

```
                 Logo-Manager
               Gegevens-bestand

                    Menu :

            1.een bestand beginnen
            2.een bestand veranderen
            3.een bestand kiezen
            4.een bestand laden
            5.een bestand opslaan
            6.opgave bestanden
            7.gegevens toevoegen
            8.gegevens veranderen
            9.gegevens bekijken
           10.gegevens sorteren
           11.gegevens tellen
           12.gegevens afdrukken

   Maak een keus door een getal (1-12) in te toetsen
```

Zo zou een menu eruit kunnen zien. We kunnen nu met de opdracht MAKE "KEUS RC, door een keus te maken uit het menu, bijvoorbeeld een van de volgende subprocedures aanroepen:

1. START - een bestand beginnen we met deze procedure.
   Functies:
   - variabelen in het werkgeheugen wissen met de procedure setup;
   - vragen om een naam voor dit bestand
   - deze naam toekennen aan de informatie die wordt opgeslagen in de variabele :feiten;
   - teruggaan naar het menu, bijvoorbeeld met de RETURN-toets.
2. VERANDER een bestand met een procedure die
   - de oude variabelenaam wist (ERN);
   - aan :feiten een nieuwe naam toekent.
3. KIES een werkbestand door:
   - uit een overzicht van alle bestanden er een te (laten) kiezen;
   - door te verwijzen naar de LAAD-optie.

   Pas op: met het kiezen van een bestand kan er nog niet gewerkt

worden.

4. LAAD een werkbestand op dezelfde manier als in LOGOMEMO. Denk eraan:
   - SETREAD "D: te gebruiken;
   - aan de variabele :feiten de zojuist ingevoerde gegevens te koppelen;
   - SETREAD [ ] af te sluiten;
   - terug te gaan naar het menu.
5. SAVEB, een bestand opslaan, gaat op dezelfde manier als in LOGOMEMO. Nu echter bestaat het bestand dat opgeslagen moet worden uit meerdere gegevens. Gebruik zoveel mogelijk de opdrachten uit LOGOMEMO.
6. CATALOGISEREN is een procedure die een opgave doet van de bestanden in de database; hij moet de opdracht CATALOG "D: bevatten.
   Een opgave van de bestanden kan ook afgedrukt worden. Maak hiervoor eerst een koppeling met de printer met SETWRITE "P: Alles wat nu op het scherm verschijnt, dus na CATALOG "D: ook de inhoudsopgave, wordt keurig afgedrukt.
   Opmerking. Geef aan het einde wel de opdracht SETWRITE [ ]!
7. GEGTOE kan de procedure zijn die de gegevens toevoegt aan het gekozen bestand. Maak hierbij gebruik van de procedure :feit. Vergeet niet uit te leggen hoe een en ander in zijn werk gaat. Vooral het gebruik van [, : en " vereist enige oefening.
8. GEGVER kan de procedure zijn die een te kiezen element (:feit) uit de lijst :feiten wist (met ERN) en vervangt door een ander :feit.
9. GEGZIE zou u de procedure kunnen noemen die alle gegevens op het scherm toont. Maak hiervoor gebruik van de procedure printfeiten. Ga weer terug naar het menu, bijvoorbeeld met de opdracht KEYP.
10. SORTEER zou een vrij omvangrijke procedure moeten zijn waarin de procedures sorteer, sort, in, voor en ssort een plaats krijgen.
11. GEGTEL kan een heel korte procedure zijn, uitgaande van tel. Geef wel aan dat na tel de procedurenaam genoemd moet worden.
12. GEGPRNT, de laatste procedure, kan geheel geschreven worden aan de hand van de procedure PRNT uit LOGOMEMO.

Opmerking. Houd bij de opzet van deze Logo-Manager het beschikbare geheugen wel in het oog. Bij LCSI Logo van Atari is de nog beschikbare geheugenruimte steeds op te vragen met PR NODES.

Op deze manier, en met behulp van de gegeven routines en hulpprocedures, kan iedereen zijn eigen Logo-Manager schrijven, een eigen database in Logo, naar eigen inzichten en ideeën. Veel succes ermee!!

3.11 Het maken van adventures (avonturenspelletjes)



Met Logo zijn op eenvoudige wijze heel aardige avonturenspelletjes te maken, spelen waarin de speler als het ware de hoofdrol speelt en zo zelf de loop van het verhaal bepaalt.

Als voorbeeld volgt nu een klein gedeelte uit 'Het grote Yama Avontuur', een avonturenspel speciaal voor kinderen geschreven. Zoals u ziet wordt niet alleen van Logo-taal gebruik gemaakt, maar ook van Logo-schildpadwereld (Turtle Graphics), Logo-sproken en Logo-muziek.

Bekijkt u de listing maar eens rustig.

```
TO BOS
CS CT SS SETBG 97
PUTSH 4 :MAAN
PR [Even wachten ....]
PR [Dan laat ik je het bos zien]
WRAP
REPEAT 2 [BOMEN]
GRAS 100 CT
PR [Wat een eng bos ,vind je niet ?]
PR [Wil je er nog steeds doorheen ?]
PR [Wat wil je ?]
PR [1.Door het bos]
PR [2.Terug naar het strand]
MAKE "KEUS RC
IF :KEUS = "2 [STRAND]
```

```
CT
PR ( SE [Goed ,] :NAAM "dan "gaan "we "verder
PR [Daar lopen we in het bos]
PR [En het wordt steeds donkerder]
SETBG 5 WAIT 60 SETBG 4 WAIT 60
SETBG 3 WAIT 60 SETBG 2 WAIT 60
SETBG 1 WAIT 60 SETBG 0
MAAN CT
PR [En nu is het helemaal donker]
PR [Wel wat eng he]
PR [En gaat het nog onweren ook]
BLIKSEM
WAIT 300 CT
PR [Plotseling .......]
PR [Wat is dat ????]
PR [Wat komt daar op ons af ?]
WAIT 200 FS CT CS UIL
SS PR [Ik ben Cas , de toveruil]
PR ( SE [Wat doe je hier ,] :NAAM "?
PR [Je mag doorlopen als je]
PR [op dit raadsel een antwoord weet] WAIT 600 CT
PR [Het heeft 4 poten maar kan niet lopen]
PR [Ra , ra wat is dat ?]
PR [1.Een paard]
PR [2.Een tafel]
PR [3.Een vis]
MAKE "ANTWOORD RC
IF :ANTWOORD = "2 [PR [Okee ga maar verder] [PR [J
ammer dat was fout , nu is je avontuur afgelopen]]
]
PR ( SE [Daag ,] :NAAM
END
```

Wilt u zelf een avonturenspel maken? Dan is het verstandig als volgt te werk te gaan.

- Ontwerp eerst een verhaal op papier en geef daarin alle, door de speler te kiezen, mogelijkheden aan. Een mogelijk schema ziet u op de volgende bladzijde.

U ziet dat er nogal wat mogelijkheden zijn. Elk van deze mogelijkheden kan echter door de speler worden gekozen. Elke mogelijkheid zal dus in een aparte procedure moeten staan.

Begin na het ontwerpen van het verhaal met het schrijven van de subprocedures. Test alle procedures zorgvuldig.

Voor een kasteel ga je ...
1. naar binnen
2. buitenom

Binnen ga je ...
1. de gang door
2. de trap op

Buiten ga je ...
1. door het kelderraam
2. verder

In de gang ga je ...
1. linksaf
2. rechtsaf

Op de trap lopend ga je ...
1. op het geroep af
2. bij het geroep vandaan

In de kelder aangekomen ga je ...
1. door de kelderdeur
2. even uitrusten

Verder lopend ga je ...
1. over de muur de kasteeltuin binnen
2. het kasteel voorbij

enzovoort, enzovoort.

- Houd, indien het avonturenspel in één keer ingevoerd wordt, rekening met de beschikbare geheugenruimte. Het is raadzamer eerst een avonturenspel op diskette te maken, waarbij steeds een gedeelte van de procedure wordt ingevoerd.
- Maak bij het schrijven van de procedures niet alleen gebruik van Logo-taal. Ook Logo-schildpadwereld, Logo-muziek en Logo-sproken kunnen zeer goed worden toegepast.
- Ontwerp ook procedures die u vaker kunt gebruiken. Denk bijvoorbeeld aan procedures als BLIKSEM, REGEN, WIND.
- Denk eraan dat ook een kort avontuur al veel procedures kost. Maak de procedures daarom zo efficiënt mogelijk.
- Gebruik variabelen met mate, en wis steeds alle niet meer gebruikte variabelen. Juist deze variabelen nemen veel geheugenruimte in beslag. Benut de geheugenruimte optimaal.

Met enige creativiteit zijn er met Logo bijzonder aardige 'avonturen' te schrijven.

3.12 Het gebruik van lijsten

Ook buiten de Logo-taalwereld kunnen we natuurlijk het fenomeen 'lijst' goed gebruiken. Laten we maar eens naar een paar voorbeelden gaan kijken.

We definiëren zelf een paar opdrachten, bijvoorbeeld SRECORD en REPLAY. Met behulp van deze opdrachten kunnen we verschillende tekeningen in het geheugen opnemen en deze later weer laten tekenen. Elke tekening kunnen we een eigen naam geven en deze later weer laten 'hertekenen' met de opdracht REPLAY :naam. Met enige aanpassing kunt u hier een uitgebreid tekenprogramma mee schrijven, compleet met SAVE- en LOAD-procedures (zie hiervoor bijvoorbeeld LOGOMEMO).

SRECORD en REPLAY zijn echter ook voor vele andere zaken te gebruiken. Wat denkt u van muziek? Als we in plaats van Turtle-opdrachten nu eens muziekopdrachten gebruikten, dan kunnen we onze eigen Logo-muziek opnemen, ieder muziekstuk een naam geven en met REPLAY :naam het gekozen muziekstuk beluisteren.

De mogelijkheden zijn legio, maar u krijgt ze niet op een presenteerblaadje.

De procedures RECORD, SRECORD, REPLAY en RW kunt u naar uw eigen smaak en idee aanpassen, veranderen en gebruiken in eigen programma's. Zoals u ziet doet de procedure RECORD niets anders dan iedere opdracht met behulp van LPUT aan het einde van een opdrachtenlijst plaatsen. Op deze manier ontstaat een opdrachtenlijst die door REPLAY opdracht voor opdracht wordt afgewerkt.

Het bovenstaande kunt u toch wel als een voortreffelijke illustratie van de gebruikersvriendelijkheid van Logo beschouwen.

Hier volgen de procedures:

```
TO SRECORD
CT SS PR [Geef een naam :]
MAKE "NAAM RW
MAKE "OPDRACHTEN []
RECORD
MAKE THING "NAAM :OPDRACHTEN
END


TO RECORD
MAKE "OPDRACHT RL
IF :OPDRACHT = "STOP [STOP]
RUN :OPDRACHT
MAKE "OPDRACHTEN LPUT :OPDRACHT :OPDRACHTEN
RECORD
END
```

```
TO REPLAY :OPDRACHTEN
IF EMPTYP :OPDRACHTEN [STOP]
RUN FIRST :OPDRACHTEN
REPLAY BF :OPDRACHTEN
END
```

```
TO RW
OUTPUT FIRST RL
END
```

3.13 Logo-tekst, de bouwstenen voor een tekstverwerker

We moeten goed onder ogen zien dat het met behulp van Logo niet mogelijk is een tekstverwerker als bijvoorbeeld WordStar te maken. Daarvoor is de taal Logo niet geschikt. In dit hoofdstuk krijgt u evenmin een complete tekstverwerker in Logo.

Wat u wel krijgt zijn de bouwstenen waarmee u zelf, op betrekkelijk eenvoudig wijze, een tekstverwerker in elkaar kunt zetten. Een tekstverwerker geheel naar uw eigen wensen en verlangens. Een tekstverwerker op maat.

Laten we maar eens gaan kijken welke 'bouwstenen' we nodig hebben.

Een eenvoudige tekstverwerker moet de volgende mogelijkheden hebben:

- een menu, dat te allen tijde kan worden aangesproken;
- een 'schrijfgedeelte', waaruit we snel weer met één enkele toets naar het menu kunnen komen;
- een 'leesgedeelte', waar we de zojuist via het toetsenbord of vanaf diskette ingevoerde tekst kunnen nalezen;
- een 'overzicht' van de reeds op diskette weggeschreven teksten;
- een 'wisgedeelte' om een tekst uit het geheugen/van diskette te wissen;
- een 'laadgedeelte' om een tekst van diskette (eventueel band) te laden;
- een 'opslaggedeelte' om teksten weg te schrijven naar diskette/band;
- een eenvoudig 'opmaakgedeelte' met bijvoorbeeld een 'SEARCH/REPLACE?-functie;
- en alles wat u verder belangrijk vindt.

Het menu

Een menu maken is niet zo ingewikkeld. Zorg voor een rustige schermopbouw en een logische indeling. Denk er ook aan steeds de niet meer gebruikte variabelen te wissen. Dit is belangrijk indien u nog geheugenruimte wilt overhouden om te kunnen schrijven.

Het is ook belangrijk dat u dit menu te allen tijde kunt aanroepen. Dat kan natuurlijk door steeds MENU in te toetsen en de procedure daarmee op te roepen. Handig is dit echter niet.

Het is beter gebruik te maken van een hulpprocedure als TERUG. Bij iedere optie van het menu plaatst TERUG de tekst

```
Toets ESC voor MENU
```

onderaan het scherm. Een druk op de ESC-toets is dan voldoende om weer in het menu te komen.

Er is gekozen voor de ESC-toets omdat deze bij het schrijven van een tekst niet wordt gebruikt; zo kan er geen verwarring ontstaan.

Een menu en een procedure TERUG kunnen er zo uitzien:

```
TO MENU
CT TS SETBG 0 ERN [KEUS VERDER]
SETCURSOR [4 1] PR [. . L o g o t e k s t . .]
SETCURSOR [4 5] PR [1.Schrijf tekst]
SETCURSOR [4 7] PR [2.Lees tekst]
SETCURSOR [4 9] PR [3.Overzicht]
SETCURSOR [4 11] PR [4.Afdrukken tekst]
SETCURSOR [4 13] PR [5.Wis tekst]
SETCURSOR [4 15] PR [6.Laadt tekst]
SETCURSOR [4 17] PR [7.Opslaan tekst]
SETCURSOR [4 20] PR [Doe je keus maar ...]
MAKE "KEUS RC
IF :KEUS = "1 [ERN :KEUS SSC]
IF :KEUS = "2 [ERN :KEUS LLE]
IF :KEUS = "3 [ERN :KEUS ZIE]
IF :KEUS = "4 [ERN :KEUS PRNT]
IF :KEUS = "5 [ERN :KEUS WIS]
IF :KEUS = "6 [ERN :KEUS LAAD]
IF :KEUS = "7 [ERN :KEUS WSCHRIJF]
END
```

```
TO TERUG
SETCURSOR [10 23] PR [Toets ESC voor MENU]
MAKE "keuze RC
IF :keuze = CHAR 27 [ERN "keuze MENU]
END
```

Schrijf

Een schrijfgedeelte zou in principe kunnen zijn:

- Begin met een variabele :tekst.
- Geef deze variabele de waarde van een lege lijst: MAKE "tekst [ ].
- Ga dan naar een recursieve subprocedure waarin de variabele :regel is opgenomen.
- Laat :regel de waarde krijgen van de ingetypte zin: MAKE "regel RL.
- Plaats :regel (afgesloten door een RETURN) achter in :tekst. Zo wordt iedere :regel een element van :tekst.
- Iedere :regel wordt achter de voorgaande in :tekst geplaatst.
- Verlaat de recursieve procedure bijvoorbeeld door het typen van een lege regel.
- Geef de mogelijkheid terug naar het menu te gaan met de procedure TERUG.

Het hoeft natuurlijk niet, maar u zou het zo kunnen doen:

```
TO SSC
CT TS
MAKE "tekst []
SCHRIJF
TERUG
END


TO SCHRIJF
TS
MAKE "regel RL
IF EMPTYP :regel [STOP]
MAKE "tekst LPUT :regel :tekst
ERN "regel
SCHRIJF
END
```

Hiervoor heeft u beslist twee procedures nodig, want de opdracht MAKE "tekst [ ] kan niet in de recursieve procedure staan en evenmin in het menu.

Terwijl u aan het schrijven bent kunt u gebruik maken van alle opmaakfuncties van het toetsenbord: CLEAR, INSERT, DELETE BACK SPACE, enzovoort.

Lees

Ook het lezen van de tekst kan het beste met twee procedures gebeuren. De eerste procedure controleert of er wel een tekst in het geheugen zit. Het is zeer belangrijk om steeds zulke controle-opdrachten in te bouwen.

Zo zouden de bouwstenen voor het leesgedeelte er uit kunnen zien:

```
TO LLE
CT
IF NOT NAMEP "tekst [TERUG]
LEES :tekst
TERUG
END


TO LEES :tekst
IF EMPTYP :tekst [STOP]
PR FIRST :tekst
LEES BF :tekst
END
```

Ziet u dat ook hier weer besloten wordt met de procedure TERUG?

Overzicht

Als we even willen kijken welke teksten er al op diskette staan, kan dat natuurlijk zeer snel gebeuren met de opdracht CATALOG "D: . U zou er de volgende procedure van kunnen maken:

```
TO ZIE
CT TS
PR [Deze staan op diskette :]
PR []
CATALOG "D:
TERUG
END
```

Afdrukken

Nog zo'n eenvoudige bouwsteen in de tekstverwerker. We maken hier gebruik van de opdracht SETWRITE "P: . Kijkt u maar:

```
TO PRNT
CT TS
SETWRITE "P:
LEES :tekst  →  →  →  →  →  SETWRITE []
TERUG
END
```

Wellicht is het verstandig ook hier een controle in te bouwen, bijvoorbeeld de opdracht:

```
IF NOT NAMEP "tekst [TERUG]
```

Zoals u weet controleert deze opdracht of de variabele :tekst wel een waarde heeft. Met andere woorden: of er wel een tekst in het geheugen zit.

Wis

Het kan voorkomen dat u een bepaalde tekst in zijn geheel uit het geheugen wilt wissen. U zou dit kunnen doen met de volgende eenvoudige procedure:

```
TO WIS
CT TS
IF NOT NAMEP "tekst [TERUG]
PR [Deze tekst wordt gewist :]
LEES :tekst
ERN "tekst
TERUG
END
```

Wilt u ook de mogelijkheid hebben een tekst van de diskette te wissen? Dit kan met de opdracht ERF "D:bestandsnaam. Verander de procedure dan zo:

1. Stel de vraag: wissen uit Geheugen / van Diskette (G / D).
2. Toets het antwoord MAKE "antwoord RC in.
3. Als het antwoord G is, gebruik dan de reeds genoemde opdrachten: IF :antwoord = "G [PR [Deze tekst wordt gewist :] LEES :tekst ERN "tekst TERUG].
4. Als het antwoord D is, vraag dan welke tekst gewist moet worden: PR [Welke tekst ?] MAKE "k RW.
5. Geef dan de opdracht de bedoelde tekst van diskette te wissen: ERF WORD "D: :k .
6. Vergeet niet de procedure TERUG aan te roepen.

Bij het vragen naar de naam van de te wissen tekst zou u een paar tekstopdrachten kunnen 'inbouwen' om te controleren of

- er wel iets is ingetypt: IF EMPTYP :k [WIS];
- de ingetypte naam niet langer is dan acht tekens: IF COUNT :k > 8 [WIS];
- de gewenste tekst wel voorkomt op de diskette.

Voor deze laatste test hebt u echter bijna altijd een extra procedure nodig. U moet dan opdrachten geven om

- alle namen van tekstbestanden in een lijst te plaatsen;
- met MEMBERP te kijken of de ingetypte naam hier ook in voorkomt.

Een heel werk dus. De procedures geven we u niet. U krijgt per slot van rekening niet alles cadeau...!

Laad / Opmaak

Een tekst laden gaat op dezelfde manier als in LogoMemo staat aangegeven. Het is weinig zinvol dat hier te herhalen. De opdracht hiervoor is SETREAD.

De procedure die u moet gebruiken kan ook veel lijken op de procedure om een tekst op te slaan. Kijk maar:

```
TO WSCHRIJF
CT TS
IF NOT NAMEP "tekst [PR [Er is geen tekst om op te
 slaan !] TERUG]
PR ( SE [De tekst] :naam "wordt "nu "opgeslagen.
SETWRITE WORD "D: :naam
LEES :tekst
SETWRITE [] TERUG
END

TO RW
OP FIRST RL
END
```

Ziet u de controle-opdracht? Hij zou natuurlijk niet nodig moeten zijn, maar ja, je weet maar nooit.

Opmaak

Het werkt plezierig wanneer u voor het opmaakgedeelte een apart klein menu maakt. Dit zou er zo uit kunnen zien:

```
L o g o t e k s t

  Opmaak - menu

  Maak uw keuze :

1.Zoek / vervang
2.Bestanden samenvoegen
3.Bestanden splitsen
4......
5......
```

Enzovoort. Hier kunt u alle hulpmiddelen kwijt die u denkt bij het schrijven van een brief ooit nodig te hebben.

Iedere keuze van het menu moet verwijzen naar een procedure. Zo zou u bij zoek/vervang kunnen verwijzen naar de later in dit hoofdstuk beschreven procedure REPLACE.

We zijn aan het einde van de bouwstenen gekomen. We hebben ze lang niet allemaal genoemd. Zo is er bijvoorbeeld de mogelijkheid om met meerdere teksten tegelijk in het geheugen te werken. Maar het is niet goed u alle plezier in het experimenteren te ontnemen. Het is toch juist veel leuker dat u, al experimenterend met de hierboven aangedragen bouwstenen, uw eigen tekstverwerker op maat maakt dan dat u een kant-en-klare tekstverwerker in de schoot geworpen krijgt.

3.14 Enige nuttige Taal-procedures

Lijsten kunnen we op velerlei manieren manipuleren, daar hebben we in dit hoofdstuk al verscheidene voorbeelden van gezien. Toch bestaat de mogelijkheid, dat u bij het programmeren van bijvoorbeeld een database net die opdrachten nodig hebt die de door u gebruikte Logo niet bezit.

Zo kan het voorkomen dat u zoekt naar een opdracht die twee lijsten samenvoegt, zodanig dat er in het resultaat geen dubbele elementen voorkomen. Het is betrekkelijk eenvoudig (met de opdracht LIST) om met

[A B C] en [C D E]

de lijst

[A B C C D E]

te verkrijgen, maar dat is niet uw bedoeling. In een Logo-listing voor Terrapin Logo (een door Apple gebruikte Logo-versie) kunt u de procedure UNION vinden, die natuurlijk wel in LCSI Logo moet worden omgezet. Hieronder vindt u de listing van deze nieuwe procedure, die de naam TOGETHER zou kunnen dragen. U krijgt de listing eerst voor Engelstalige Logo, daaronder voor Nederlandstalige Logo.

```
TO TOGETHER :LIJST1 :LIJST2
IF EMPTYP :LIJST1 [OP :LIJST2]
IF MEMBERP ( FIRST :LIJST1 ) :LIJST2 [OP
 TOGETHER ( BF :LIJST1 ) :LIJST2] [OP FP
UT ( FIRST :LIJST1 ) TOGETHER ( BF :LIJS
T1 ) :LIJST2]
END
```

Willen we nu de lijsten [A B C D E F G] en [A B E F X Z] samenvoegen, dan toetsen we in:

```
PR TOGETHER [A B C D E F G] [A B E F X Z]
```

en we krijgen als uitkomst: [A B C D E F G X Z].

Opmerking. Vergeet niet de opdracht PR voor TOGETHER op te nemen (tenzij u TOGETHER als subprocedure gebruikt). Zonder PR weet Logo namelijk niet wat het met de nieuwe lijst moet doen en krijgen we de boodschap:

```
YOU DON'T SAY WHAT TO DO WITH [A B C D E F G X Z]
```

We hebben nu een procedure die twee lijsten samenvoegt. Deze weg vervolgend kunnen we natuurlijk een procedure schrijven die twee lijsten vergelijkt; een procedure die ziet of er in twee verschillende lijsten wellicht overeenkomstige elementen zitten. Zo'n procedure kan o.a. in bestandsprogramma's zeer bruikbaar zijn.

```
TO COMPARE :LIJST1 :LIJST2
IF EMPTYP :LIJST1 [OP []]
IF MEMBERP FIRST :LIJST1 :LIJST2 [OP FPU
T FIRST :LIJST1 COMPARE BF :LIJST1 :LIJS
T2] [OP COMPARE BF :LIJST1 :LIJST2]
END
```

Een voorbeeldje:

```
PR COMPARE [A B C] [B C D]
B C
```

Even eenvoudig als doeltreffend.

Maar we zijn nog niet klaar. Soms is het nodig een element uit een lijst te vervangen door een nieuw element. Denk bijvoorbeeld maar even aan een gegevensbestand waar oude gegevens door nieuwe moeten worden vervangen. In zo'n geval zou het gemakkelijk zijn als er een opdracht REPLACE bestond die het ene element in een lijst door het andere kon vervangen.

Welnu, sommige Logo-versies beschikken inderdaad over de opdracht REPLACE, maar we kunnen hem ook eenvoudig zelf maken:

```
TO REPLACE :O :N :LIJST
IF EMPTYP :LIJST [OP []]
IF :O = FIRST :LIJST [OP FPUT :N REPLACE
 :O :N BF :LIJST] [OP FPUT FIRST :LIJST
REPLACE :O :N BF :LIJST]
END
```

Zo wordt `PR REPLACE "A "B [A C D] ... [B C D]`

Een laatste hulpprocedure is de opdracht CHECK. CHECK kijkt voor u naar de waarde van een variabele en drukt die af. Dit is de, zeer eenvoudige, procedure:

```
EDIT "CHECK :a
PR (SE :a "heeft "als "waarde THING :a
END
```

We zouden ook gewoon alleen de opdracht THING kunnen gebruiken:

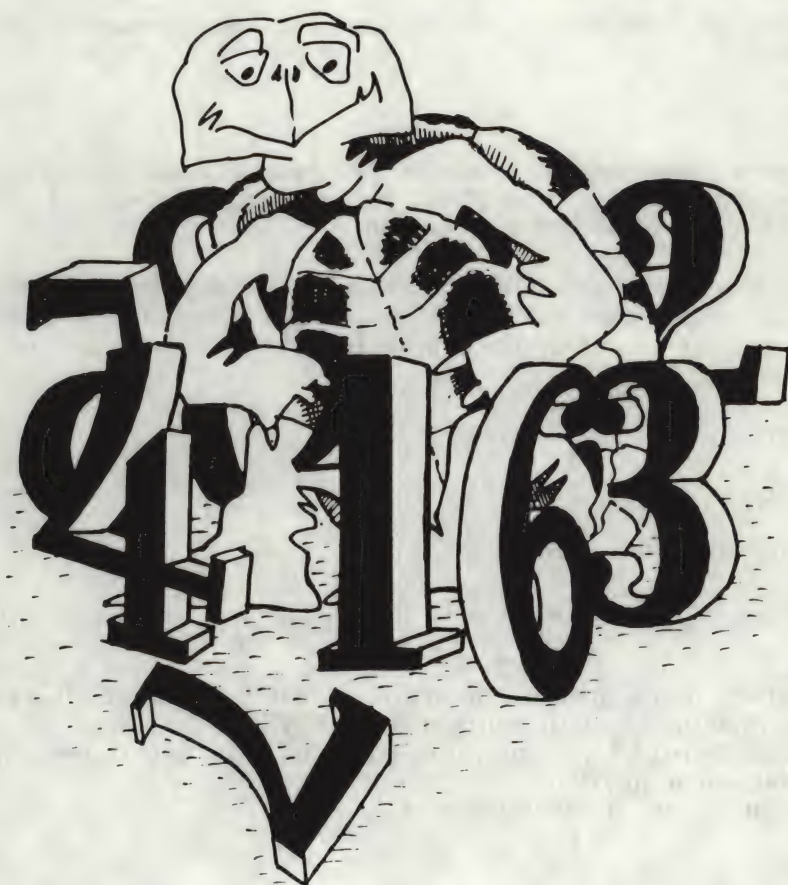
```
PR THING :a
```

Dit heeft hetzelfde resultaat.

Op deze manier zijn er vele nuttige en bovenal leerzame procedures te schrijven die het programmeren gemakkelijker maken.

Hoofdstuk 4

LOGO: CIJFERS EN GETALLEN



4.1 Inleiding

Zoals wij in het voorwoord reeds hebben gezegd, kunnen computers eigenlijk alleen maar 'cijfertaal' verstaan, en wordt iedere opdracht die we de computer geven ogenblikkelijk omgezet in een reeks enen en nullen. We spreken hierbij van BITs (BInary digiTs). Straks komen we daar uitgebreid op terug.

We kunnen in Logo ook rekenkundige bewerkingen uitvoeren. In dit hoofdstuk zult u er verscheidene zien.

Logo accepteert zowel geheeltallige (integers) als decimale getallen. Voor alle duidelijkheid:

645	is een geheel getal	en
6.45	is een decimaal getal.	

Soms zien we, bijvoorbeeld als uitkomst van een berekening, het volgende:

3E6

Dit is de wetenschappelijke notatie die Logo gebruikt bij decimale getallen van meer dan zes cijfers. 3E6 wil zeggen: 3 maal 10 (tot de macht 6) → 3 x 1.000.000 = 3.000.000.

De basisbewerkingen optellen, aftrekken, vermenigvuldigen en delen zijn mogelijk als INFIX-notatie. De INFIX-notatie gaat ervan uit dat het rekenkundige bewerkingsteken (+ - * /) tussen de getallen staat:

```
Optellen          : PRINT 5 + 4
                    9
Aftrekken         : PRINT 5 - 4
                    1
Vermenigvuldigen  : PRINT 5 * 4
                    20
Delen             : PRINT 5 / 4
                    1.25
```

Opmerking. Denk goed om de spaties; PRINT 5/4 geeft bij sommige Logo-versies niet dezelfde uitkomst als PRINT 5 / 4.
Het -teken kan (zonder spatie) ook worden gebruikt om een negatief getal aan te geven.

```
Voorbeeld         : PRINT 5 + -4
                    1
```

Voor het optellen en vermenigvuldigen kunnen we echter ook een PREFIX-notatie gebruiken in combinatie met SUM en PRODUCT. De bewerkingen SUM en PRODUCT staan hierbij *voor* de getallen die respectievelijk opgeteld en vermenigvuldigd moeten worden.

Kijk maar:

```
PRINT SUM 5 4
9
PRINT SUM (2 5 7 6 9 4 -3)
30
PRINT PRODUCT 5 4
20
PRINT PRODUCT (5 4 5)
100
```

4.2 Wiskundige functies

In deze paragraaf kijken we naar een aantal wiskundige functies.

SIN Hiermee wordt de sinus van een hoek berekend, bijvoorbeeld:

```
PRINT SIN 100
0.9848
```

COS Hiermee berekenen we de cosinus van een hoek, bijvoorbeeld:

```
PRINT COS 30
0.86605
```

TAN Op basis van de sinus- en cosinus-berekening kunnen we ook de tangens van een hoek berekenen. Sommige Logo-versies bevatten de TAN-functie standaard, maar de functie kan ook door een procedure gedefinieerd worden:

```
EDIT "TAN :HOEK
OUTPUT (SIN :HOEK) / COS :HOEK
END
```

Een voorbeeld:

```
PRINT TAN 444
9.51511672
```

SQRT Hiermee wordt de vierkantswortel (SQare RooT) van een getal berekend:

```
PRINT SQRT 100
10
```

REMAINDER Dit is een heel speciale functie. De opdracht vraagt om twee getallen, deelt het eerste door het tweede en geeft als uitkomst de rest van de deling. Kijk maar:

```
PRINT REMAINDER 19 5
4
PRINT REMAINDER 19 50
19
```

Hiermee kunnen we een procedure maken die kijkt of een getal deelbaar is door 3:

```
EDIT "DEEL :n
OUTPUT 0 = REMAINDER :n 3
END
```

Dit is het gevolg:

```
DEEL 3807
TRUE
DEEL 998
FALSE
```

Er zijn ook Logo-opdrachten om getallen af te ronden:

INT maakt van een getal een geheel getal door de decimalen te verwijderen:

```
PRINT INT 6.2343
6
PRINT INT SQRT 55
7
```

ROUND rondt een getal af naar de dichtstbijzijnde gehele waarde:

```
PRINT ROUND 6.12345
6
PRINT ROUND 6.50123
7
```

ABS Sommige Logo-versies bezitten ook nog een ABS-functie (o.a. Atari 520 ST Logo). De ABS-functie geeft de absolute waarde van een getal.
Zoals bijna met iedere Logo-opdracht het geval is, kunnen we ook de opdracht ABS zelf maken. Daarvoor gebruiken we de volgende procedure:

```
EDIT "ABS :n
OP IF :n < 0 [-:n] [:n]
END
```

OP IF :n < 0 [-:n] [:n] wil zeggen:
als :n kleiner is dan 0, geef dan als uitvoer -:n, anders (dus als :n groter dan of gelijk aan 0 is) geef :n.

In andere Logo-versies moet zo'n ALS ... DAN ... ANDERS-opdracht (IF ... ELSE ... THEN) als volgt geschreven worden:

```
IF :n < 0 THEN OP -:n ELSE OP :n
```

We zagen ze al eerder, de tekens < (kleiner dan) en > (groter dan). Ze worden steeds in de INFIX-notatie gebruikt, dus altijd tussen twee waarden.

4.3 Talstelsels

Dan gaan we nu kijken naar wat we talstelsels noemen. In ons huis-tuin-en-keukenrekenen maken we steeds gebruik van het tientallig (decimale) stelsel. Dat wil zeggen dat we ieder getal kunnen herleiden tot een optelling van machten van 10. Zo kunnen we het getal 826 schrijven als:

8 * 10 tot de macht 2 +
2 * 10 tot de macht 1 +
6 * 10 tot de macht 0

Ofwel: 8 * 100 + 2 * 10 + 6 * 1.

Binnen het decimale stelsel splitsen we een getal dus op in eenheden, tientallen, honderdtallen, duizendtallen, enzovoort.

Nu is het decimale stelsel echter niet het enige talstelsel. We kunnen elk ...tallig stelsel gebruiken. Het achttallig stelsel bijvoorbeeld werkt niet met machten van 10, maar van 8. Willen we dus een decimaal getal in het achttallig stelsel schrijven, dan moeten we de machten van 8 kennen.

Het decimale getal 75 kunnen we dan in het achttalig stelsel zetten:

75 is groter dan 64 (8 tot de macht 2) maar kleiner dan 512 (8 tot de macht 3) dus:

75 = 1 * 64 (8 tot de macht 2) + 11 en
11 = 1 * 8 (8 tot de macht 1) + 3 en
3 = 3 * 1 (8 tot de macht 0).

Aldus wordt het getal 75 in het achttallig stelsel uitgesproken als 113.

Omdat we ieder getal kunnen herleiden tot een optelling van machten van 8, zult u de cijfers 8 en 9 in het achttallig stelsel niet tegenkomen. Het cijfer 7 is het grootste cijfer in dit talstelsel.

Nog een voorbeeld:
Om het getal 1001 in het achttallig stelsel te schrijven gaan we als volgt te werk:

De machten van 8 zijn: 1, 8, 64, 512, 4096

1001 = 1 * 512	(8 tot de macht 3)	+	489	en
489 = 7 * 64	(8 tot de macht 2)	+	41	en
41 = 5 * 8	(8 tot de macht 1)	+	1	en
1 = 1 * 1	(8 tot de macht 0)			

Zo is het decimale getal 1001 in het achttallig stelsel 1751.

Een ander talstelsel is het tweetallig (ook wel binaire) stelsel. Het tweetallig stelsel werkt met machten van 2: 1 2 4 8 16 32 64 128, enzovoort. Zo is 54 in het tweetallig stelsel:

54 = 1 * 32	(2 tot de macht 5)	+	22	en
22 = 1 * 16	(2 tot de macht 4)	+	6	en
6 = 0 * 8	(2 tot de macht 3)	+	6	en
6 = 1 * 4	(2 tot de macht 2)	+	2	en
2 = 1 * 2	(2 tot de macht 1)	+	0	en
0 = 0 * 1	(2 tot de macht 0)			

Kijk maar:

32	16	8	4	2	1
1	1	0	1	1	0

Dus wordt 54 binair als 110110 geschreven.

Zullen we binair eens tot 10 tellen?

1	=	1
2	=	10
3	=	11
4	=	100
5	=	101
6	=	110
7	=	111
8	=	1000
9	=	1001
10	=	1010

Zo kunnen we dus elk getal opschrijven als combinatie van enen en nullen. Zoals in het decimale stelsel een getal te schrijven is als een

optelling van machten van 10, zo is in het binaire stelsel een getal te schrijven als een optelling van machten van 2.

Wanneer we van het getal 3 een binair getal maken, gaan we als volgt te werk:

```
3 / 2 = 1   (en 1 onthouden)
```

Daarom zetten we een 1 onder de nulde macht van twee (de 'eenheden'):

32	16	8	4	2	1	32	16	8	4	2	1
					1	2^5	2^4	2^3	2^2	2^1	2^0

```
1 / 2 = 0   (en 1 onthouden)
```

Aldus krijgen we ook in de volgende kolom, onder de eerste macht van 2 (de 'tweetallen') een 1:

32	16	8	4	2	1
				1	1

Het decimale getal 3 wordt dus in het tweetallig stelsel als 11 geschreven, omdat 3 gelijk is aan:

```
1 * 2 tot de macht 1  :  1 * 2 = 2      en
1 * 2 tot de macht 0  :  1 * 1 = 1
```

Dit delen en onthouden van de rest hoeven we niet zelf uit te rekenen, dat kunnen we Logo ook laten doen. En wel met de volgende recursieve procedure, die we trouwens ook voor andere talstelsels dan het binaire stelsel kunnen gebruiken:

```
EDIT "BASE2  :n
IF :n = "0 [OP "]
OP WORD BASE2 (INT :n / 2) (REMAINDER INT :n 2)
END
```

Opmerking. In sommige Logo-versies moet na de IF-opdracht steeds THEN gebruikt worden. De tweede regel van de procedure wordt dan:

```
IF :n = "0 THEN OP "
```

Enkele kanttekeningen bij deze procedure

- De procedure is recursief.
- BASE2 deelt steeds het getal :n door 2, bepaalt hiervan het gehele deel (INT :n / 2) en onthoudt (in WORD) de rest (altijd 1 of 0).

Dit gebeurt met REMAINDER INT :n 2.
- Tegelijkertijd roept BASE2 zichzelf weer aan.
- Maar nu is de waarde van :n gelijk aan INT :n / 2.
- Er is gebruik gemaakt van OP (OUTPUT) in plaats van PR, omdat het binaire getal niet in een keer berekend wordt, maar langzaam wordt opgebouwd.

We kunnen van deze procedure ook een BASE8 of een BASE6 maken om respectievelijk getallen in het acht- of zestallig stelsel uit te rekenen. Let wel, dan moet ook :n door respectievelijk 8 en 6 gedeeld worden. Of we kunnen het talstelsel variabel maken:

```
EDIT "BASE :n :s
IF :n = "0 [OP "]
OP WORD BASE (INT :n / :s) :s (REMAINDER INT :n :s)
END
```

Op deze manier hebben we alle talstelsels in één procedure.

Andersom kan ook. Hier is de procedure die van een getal (in elk willekeurig stelsel) een decimaal getal maakt:

```
EDIT "BBASE :n :s
IF :n = " [OP 0]
OP (:s * BBASE BL :n :s) + (LAST :n)
END
```

De variabele :n wordt hier gehanteerd om het te bewerken getal aan te geven. De variabele :s wordt gebruikt om het talstelsel aan te geven.

Enkele voorbeelden:

```
?PR BASE 8 2
1000
?PR BASE 8 3
22
?PR BASE 8 4
20
?PR BASE 10029837 5
10031423322
?PR BASE 92837465551002 7
2536116663541001001000020000010000
?PR BASE 12 6
20
?PR BASE 88 8
130
?PR BASE 110 90
120
```

```
?PR BASE 75 8
113
?
?PR BBASE 1000 2
8
?PR BBASE 22 3
8
?PR BBASE 123456789 9
54481005
?
?
?enzovoort , enzovoort
```

4.4 Hexadecimale getallen (het zestientallig stelsel)

Zoals gezegd kan Logo worden gebruikt voor de meest uiteenlopende rekenkundige bewerkingen. Bovendien is Logo zo gebruikersvriendelijk, dat zelfs de meest ingewikkelde berekeningen in Logo betrekkelijk eenvoudig te maken zijn. Dit heeft u in het voorafgaande kunnen zien.

Als laatste wordt nu behandeld hoe bijvoorbeeld het omrekenen van decimale naar hexadecimale getallen in Logo is te definiëren.

Hexadecimale getallen worden in de computerwereld vaak gebruikt. Alle opdrachten in machinetaal bijvoorbeeld worden vaak als hexadecimale getallen geschreven. Zoals u weet zijn opdrachten in machinetaal de meest elementaire opdrachten die door de computer begrepen worden. Het hart van de computer, de CVE (Centrale VerwerkingsEenheid) wordt hier direct door aangesproken.

Ons decimale stelsel kent de getallen: 0,1,2,3,4,5,6,7,8 en 9. In het zestientallig (hexadecimale) stelsel wordt gebruik gemaakt van: 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14 en 15. In het zestientallig stelsel is dus 15 het grootste cijfer, net zoals 9 het grootste cijfer is in het tientallig stelsel.

Om de 'cijfers' 10,11,12,13,14 en 15 door één teken voor te stellen is gekozen voor de letters: A,B,C,D,E en F:

10	A
11	B
12	C
13	D
14	E
15	F

Gaan we bij het decimale stelsel uit van machten van 10,
bij het achttallig stelsel van machten van 8,
bij het tweetallig stelsel van machten van 2,

bij het hexadecimale stelsel gaan we uit van machten van 16. Dus:

16 tot de macht 0 : waarde 1 (16^0)
16 tot de macht 1 : waarde 16 (16^1)
16 tot de macht 2 : waarde 256 (16^2)
16 tot de macht 3 : waarde 4096 (16^3), enzovoort.

Zo is het hexadecimale getal 2D0:

0 * 1	=	0
D * 16 = 13 * 16	=	208
2 * 256	=	512
Dus		720 in het decimale stelsel

Nog enkele voorbeelden:

hexadecimaal 11 = decimaal 17 (11 = 1*16 + 1)
hexadecimaal 1A = decimaal 26 (1A = 1*16 + 10)
hexadecimaal A1 = decimaal 161 (A1 = 10*16 + 1)
hexadecimaal AB = decimaal 171 (AB = 10*16 + 11)
hexadecimaal 3F = decimaal 63 (3F = 3*16 + 15)

Dat valt nogal mee!

Nu kunnen we Logo ook de hexadecimale waarde van een getal laten berekenen. Hieronder volgende de procedures die we ervoor kunnen gebruiken. Voor het gemak is uitgegaan van decimale getallen tussen 0 en 255. De procedure kan echter eenvoudig worden uitgebreid. Bekijk de procedures maar eens rustig. Vooral HEX is een belangrijke procedure, omdat hierin de berekeningen plaatsvinden.

```
TO RG
OP FIRST RL
END

TO HEXA
TS CT
PR [Geef een getal tussen 0 - 255]
MAKE "D RG
HEX :D
END

TO ITEM :N :O
IF EMPTYP :O [OP "]
IF :N = 1 [OP FIRST :O]
OP ITEM :N - 1 BF :O
END
```

→

```
TO HEX :D
INIT
MAKE "A INT :D / 16
MAKE "H ITEM :A + 1 :HEXGET
MAKE "B :D - :A * 16
MAKE "L ITEM :B + 1 :HEXGET
MAKE "X WORD :H :L
MAKE "XX WORD 0 :X
PR ( SE [Het decimale getal was :] :D
PR ( SE [Het hexadecimale getal is :] :XX
PR []
PR [Nog een getal J / N ?]
MAKE "ANTW RC
IF :ANTW = "J [HEXA] [STOP]
END

TO INIT
MAKE "HEXGET [0 1 2 3 4 5 6 7 8 9 A B C D E F]
END
```

De volgende procedures zijn gebruikt:

RG
HEXA
ITEM
HEX
INIT

RG — is een zeer belangrijke hulpprocedure. Hiermee wordt het te bewerken getal gelezen. RC zou hier niet werken, want RC (Read Character) leest maar één teken, terwijl een getal meestal uit meer tekens bestaat. RL werkt evenmin, omdat RL het getal in een lijst plaatst (en een lijst is niet rekenkundig te bewerken). RG is daarom een prima alternatief, en opnieuw een illustratie van het zelf in Logo opdrachten definiëren.

HEXA — is de hoofdprocedure. Hiermee wordt de hele bewerking gestart.

ITEM — is een hulpprocedure, die we al eerder hebben besproken. ITEM is hier nodig om het juiste hexadecimale teken uit de lijst HEXGET te halen.

HEX — is de rekenkundige procedure waarvan de werking hieronder wordt uitgelegd.

INIT — is een hulpprocedure waarin de variabele :HEXGET zijn waarde krijgt. De waarde van HEXGET is een lijst met alle hexadecimale tekens.

Werking van HEX

Laten we als voorbeeld het decimale getal 42 nemen.

- HEX 42 treedt in werking. :D heeft nu de waarde 42 (in HEXA) gekregen.
- :A krijgt de waarde 2, want INT 42 / 16 is 2.
- Voor :H wordt nu een hexadecimaal teken uit HEXGET gehaald, namelijk het derde teken (2 + 1); :H krijgt hier dus de waarde 2.
- :B krijgt de waarde :D (= 42) - :A * 16 (= 32). Dit is nodig om de rest van de bewerking :D / 16 om te zetten in een hexadecimaal getal. :B krijgt dus de waarde 10.
- Voor :L wordt opnieuw een hexadecimaal teken uit HEXGET gehaald, ditmaal het elfde (10 + 1). Daarmee krijgt :L de waarde A.
- Nu moet er nog een hexadecimaal getal worden gevormd van :H en :L. Hiervoor zorgt :MAKE "X WORD :H :L.
- Om er nog een nul voor te zetten wordt tenslotte nog de variabele "XX gecreëerd.

Tenslotte worden het gegeven decimale getal en zijn hexadecimale notatie keurig afgedrukt.

Ter afsluiting van dit hoofdstuk nog een paar opmerkingen.

Natuurlijk is het handig om in Logo ook rekenkundige bewerkingen te kunnen uitvoeren. We hebben gezien dat dit bovendien op een zeer gebruikersvriendelijke manier mogelijk is. Toch zullen we de in dit hoofdstuk genoemde opdrachten en procedures niet direct voor wiskundige en/of boekhoudkundige doeleinden gaan gebruiken. Voor wiskundige en boekhoudkundige toepassingen zijn andere computertalen veel geschikter.

Toch is het aardig kennis te nemen van de rekenkundige mogelijkheden van Logo, al was het alleen maar ter illustratie van de logica van de procedures. Zelfs ogenschijnlijk grote rekenkundige problemen worden in Logo opgesplitst in kleine (dus overzichtelijke) probleempjes. De hexadecimale procedures zijn hier een goed voorbeeld van. Voor velen zit hierin de Logica van Logo.

Rekenkundige bewerkingen en rekenkundige opdrachten zullen we in dit boek regelmatig tegenkomen.

- Bij de Logo-schildpadwereld:

```
REPEAT 360 / :HOEK [FD :ZIJDE RT :HOEK]
IF REMAINDER 360 :HOEK = "0 [STOP]
```

- Bij Logo-taal:

```
IF NOT EQUALP FIRST :a FIRST :b [OP (ASCII FIRST :a)
 < (ASCII FIRST :b)]
```

- Bij Logo-muziek:

```
SETENV 0 :A TOOT 0 261 :B :C MAKE "A :A - 1
```
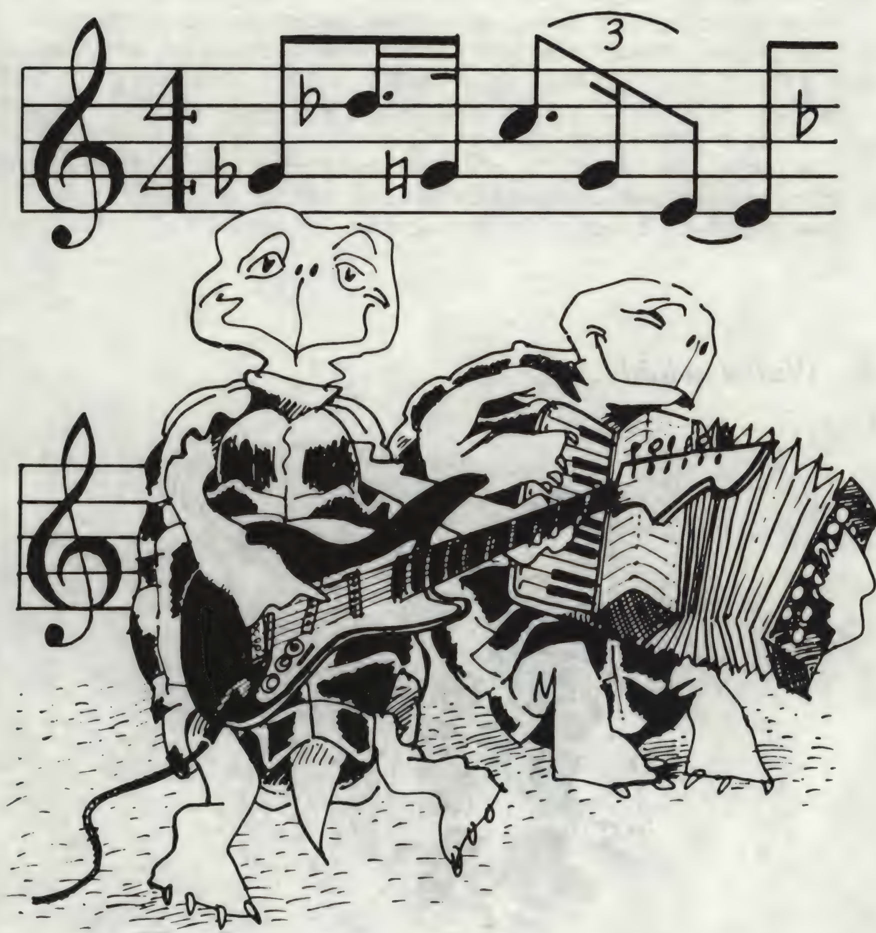
- Bij Logo-sproken:

```
EACH [SETSP (RANDOM 100) + 50]
```

Rekenkundige bewerkingen komen vaker voor dan u denkt! In alle Logo-werelden worden rekenkundige bewerkingen gebruikt, dus bestudeer dit hoofdstuk goed.

Hoofdstuk 5

DE WONDERE WERELD VAN LOGO-MUZIEK



5.1 Inleiding

De wereld van Logo-Muziek is wellicht de meest onbegrepen en minst gebruikte mogelijkheid van Logo. En dat is niet zo verwonderlijk. Vele Logo-versies bieden namelijk weinig of geen mogelijkheden daartoe.

De Atari 8-bits Logo, MSX Logo en Spectrum Logo hebben relatief weinig muziekmogelijkheden. De Atari 16-bits Logo kent helemaal geen 'muziekwereld'. De Commodore Logo's, evenals de BBC Logo's, echter geven de gebruiker vele muziekmogelijkheden.

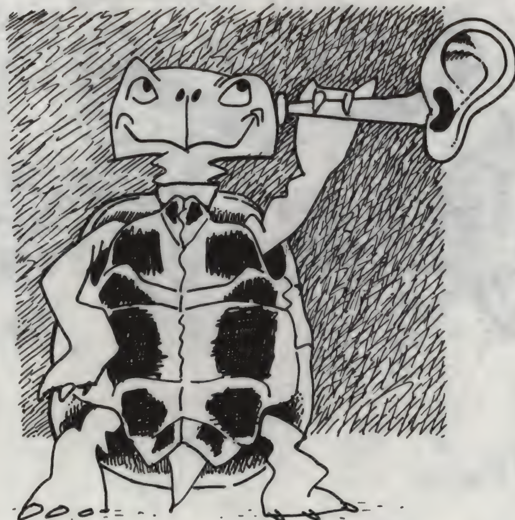
Het bovenstaande is echter geen reden om Logo-Muziek maar over te slaan. Uitgaande van de Atari LCSI Logo, die slechts twee opdrachten kent, wordt u in dit hoofdstuk getoond hoe op betrekkelijk eenvoudige wijze 'geluid' gemaakt kan worden.

Voordat we echter overgaan tot het tot stand brengen van geluiden is het zinvol eerst eens even stil te staan bij de vraag: Wat is geluid? Pas dan kunnen we onze 'eigen' geluiden gaan maken, met de Logo-hulpmiddelen (al dan niet groot in aantal) die in onze Logo-versie beschikbaar zijn.

Aan het einde van dit hoofdstuk treft u enkele praktische voorbeeldprocedures aan, alsmede de 'Turtle Music'.

Maar eerst de belangrijke vraag: Wat is geluid?

5.2 Wat is geluid?



Het beantwoorden van deze vraag is moeilijker dan u denkt. Het antwoord op deze vraag is namelijk in sterke mate afhankelijk van de persoon aan wie de vraag gesteld wordt. Een musicus, een bioloog, een natuurkundige, een muziektherapeut zullen allen de vraag beantwoorden vanuit hun eigen subjectiviteit.

Als we de waarneming door het menselijk oor buiten beschouwing laten, is geluid niets anders dan een trillingsverschijnsel:

- Een geluidsbron 'komt in beweging' en veroorzaakt een trilling.
- Deze trilling wordt overgenomen door de omliggende materie (bijvoorbeeld lucht) en plant zich zo voort.
  Dit noemen we een golfverschijnsel.
- Eventueel wordt deze trilling opgevangen door een trommelvlies, dat daardoor ook gaat trillen.
- Aldus wordt een geluid ervaren, en eventueel prettig gevonden.

Geluid kan een natuurlijke oorsprong hebben: golven, wind, blaffen, spreken. Geluid kan echter ook een mechanische oorsprong hebben: instrumenten of machines bijvoorbeeld.

Geluid, ongeacht 'waar' het vandaan komt, heeft een aantal belangrijke kenmerken:

1. De sterkte (het volume)
2. De hoogte
3. De duur
4. De klankkleur
5. Het klankverloop.

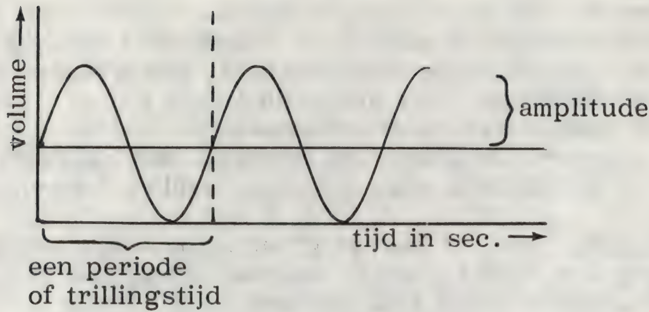
Willen we nu op mechanische of elektronische manier geluiden voortbrengen, dan zullen we de genoemde kenmerken moeten kunnen beheersen en controleren. Zoals gezegd is geluid, technisch gesproken, niet meer dan een trillingsverschijnsel.

De gevoelswaarde, dat wil zeggen of een geluid mooi, lelijk, rustgevend, verstorend of opzwepend is, wordt bepaald door het menselijk bewustzijn, en is dus een relatief begrip. De gevoelswaarde van een bepaald geluid is voor ieder mens weer anders.

Samenvattend: geluid is een fysische trilling in de tijd.
Onder dit trillingsverschijnsel verstaan we de trillingen ofwel de geluidsgolven die via de lucht ons gehoorzintuig bereiken.

Laten we de kenmerken van geluid eens bekijken:

1. De sterkte van een geluid wordt bepaald door de grootte van de geluidsgolven. De maximale uitwijking van de geluidsgolven wordt 'amplitude' genoemd. Naarmate de uitwijking groter is wordt een 'harder' geluid waargenomen. De meetwaarde voor geluidssterkte wordt decibel genoemd. Een en ander ziet u in het diagram op de volgende bladzijde.

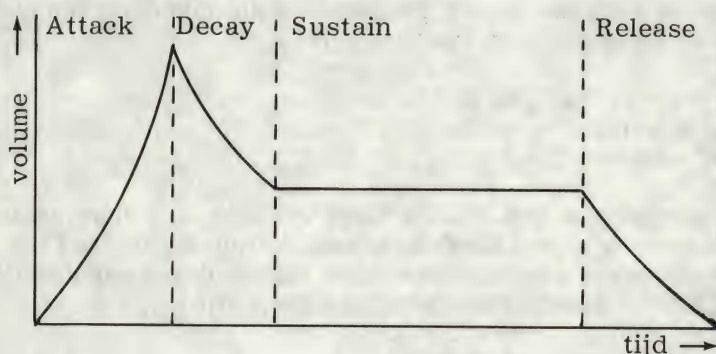


2. De hoogte van een geluid wordt bepaald door het aantal trillingen per seconde. U ziet dit in het bovenstaande diagram. We spreken hier over de frequentie van een geluid. De meetwaarde voor frequentie is hertz (Hz). Zo wordt toon A voortgebracht bij 440 trillingen per seconde. We zeggen dan dat deze toon een frequentie van 440 Hz heeft.
3. De duur van een geluid wordt bepaald door de tijdsduur waarbinnen de trillingen plaatsvinden. Hoe lang de trillingen waarneembaar zijn (het geluid hoorbaar is) hangt in sterke mate af van het volumeverloop enerzijds en van het waarnemingsvermogen van de luisteraar anderzijds.
4. De klankkleur van een geluid wordt bepaald door de vorm van de geluidsgolf. Elke golfvorm heeft een verschillende 'harmonische inhoud'. Een 'harmonische trilling' geeft een zuivere toon van één frequentie. Vaak horen we echter niet slechts deze grondfrequentie, maar tevens hogere en lagere frequenties. Onder harmonische inhoud wordt nu verstaan de verhouding van deze frequenties onderling en de verhouding tot de grondfrequentie. Zo is bij een grondtoon van 440 Hz (we noemen dit de eerste harmonische frequentie) de frequentie van 880 Hz de tweede harmonische frequentie.
   We onderscheiden de volgende golfvormen:
   - de harmonische golfvorm (elektronisch)
   - de zaagtand golfvorm (o.a. viool, hoorn)
   - de vierkants- (of blok-) golfvorm (o.a. klarinet, xylofoon)
   - de puls golfvorm (o.a. piano, hobo)
   - de driehoeksgolfvorm (o.a. fluit)

   Zoals u ziet heeft ieder instrument een eigen karakteristieke golfvorm. De namen van de golfvormen zijn afgeleid van hun grafische afbeelding.
5. Het klankverloop van een geluid wordt bepaald door de tijd waarin het volume van een geluid aanzwelt en afneemt. Het klankverloop van een geluid (in het Engels 'envelope' geheten) wordt ook wel de 'omhullende van het signaal' of 'transiënt' genoemd.
   Meestal worden in het verloop van een klank vier fasen onderscheiden:

1. Het aanzwellen van een klank tot het maximale niveau. De meest gebruikte term hiervoor is Attack.
2. Het vervallen van het volume tot het 'nagalmvolume'. We noemen dit de Decay.
3. Het nagalmen van de klank. We noemen dit de Sustain.
4. Het uitsterven van de klank. We noemen dit de Release.

Een en ander ziet u in het onderstaande diagram.



Laten we als voorbeeld eens een op de piano aangeslagen toon nemen. We onderscheiden dan:

De Attack : Bij het aanslaan van de toets wordt in zeer korte tijd een vrij hoog volumeniveau bereikt.
De Decay : Dit volume wordt echter niet vastgehouden, maar zakt terug (vervalt) tot het 'nagalmvolume'.
De Sustain : Het nu bereikte volume wordt gedurende langere of kortere tijd vastgehouden.
De Release : Tenslotte loopt het volume van de toon sterk terug totdat de toon niet langer hoorbaar is.

Iedere klank (ongeacht van welke geluidsbron) heeft een specifiek verloop. Zo verschilt het klankverloop van een toon voortgebracht op een piano nogal met het verloop van een klank voortgebracht op een orgel. Het klankverloop van een toon gespeeld op twee verschillende piano's kan zelfs nog verschillen. De bouw (van het instrument), het gebruikte materiaal, de grootte en de vorm van het instrument zijn bepalend voor het klankverloop.

Soms ligt het klankverloop (en daarmee voor een groot gedeelte ook de 'kleur' van de klank) vast in het instrument.

Een tambourijn biedt minder mogelijkheden dan een saxofoon of een gitaar. Dat komt doordat het klankverloop op de twee laatstgenoemde instrumenten beter te controleren en te beheersen is.

Indien op een elektronisch instrument als een synthesizer klan-

ken worden gemaakt, kan behalve de golfvorm ook het klankverloop worden bepaald. Met de ADSR (Attack, Decay, Sustain en Release) regeling kunnen dan zowel klanken met een 'korte' uitsterftijd (bijvoorbeeld orgel) als klanken met een 'lange' uitsterftijd (bijvoorbeeld piano) worden gemaakt. Door de karakteristieken van een klank te beheersen kan bijna ieder geluid worden geproduceerd.

De computer, met Logo als programmeertaal, heeft die mogelijkheden niet. Dat komt doordat we met Logo niet alle karakteristieken van een klank kunnen beheersen en controleren.

Laten we eens zien hoe we de genoemde geluidskenmerken vanuit Logo kunnen beheersen en controleren.

1. De sterkte van het geluid
2. De hoogte van het geluid
3. De duur van het geluid

Deze drie kenmerken zijn vanuit Logo volledig te controleren. Zij vormen de basis van het muziek-maken vanuit Logo. In §5.3 zullen deze kenmerken worden besproken en zal worden behandeld hoe ze vanuit Logo te beheersen en te controleren zijn.

4. De klankkleur van het geluid

Slechts weinig Logo-versies beschikken over de mogelijkheden om dit geluidsaspect te beheersen. Het is een moeilijke (en theoretische) materie, die maar voor een klein gedeelte van de Logo-gebruikers interessant is. Daarom wordt er hier niet verder op ingegaan. Gebruikers van bijvoorbeeld de Commodore Logo's vinden in hun handleiding voldoende aanwijzingen om hiermee te experimenteren.

5. Het klankverloop van een geluid.

Hoe dit belangrijke geluidsaspect kan worden beheerst lezen we in §5.4. Daar komen we de 'Envelope Shaper' tegen. De 'Envelope Shaper' biedt in sommige Logo-versies de mogelijkheid het klankverloop van een geluid te beheersen.

We vatten dit stukje theorie nog even kort samen. Voordat we, met de computer, een geluid willen produceren, moeten we ons afvragen:

1. Hoe hard moet het geluid zijn?
2. Hoe hoog moet het geluid zijn?
3. Het lang moet het geluid duren?
4. Hoe moet het geluid klinken?

Pas wanneer we deze factoren kunnen beheersen kunnen we met behulp van de opdrachten in Logo trachten een klank te produceren.

5.3 Muziek maken met Logo



Dit boek is weliswaar bestemd voor gebruikers van verschillende Logo-versies, maar toch gaan we in het bijzonder in op de muziekopdrachten van de Atari LCSI Logo. Hoewel in diverse Logo-versies soms andere primitieven bestaan, is het ook mogelijk dat opdrachten met gelijke werking soms anders heten (bijvoorbeeld SOUND).

De muziekwereld van Logo is de wereld van de frequenties, van notenwaarden en van de Envelope.

Atari LCSI Logo beschikt over twee muziekopdrachten. Dat is niet bepaald veel. Mensen die met Atari Logo werken zullen dan ook soms wel jaloers zijn op de mensen die met Commodore of BBC Logo werken. Toch zult u ontdekken dat zelfs met slechts twee muziekopdrachten nog erg veel muziek te maken is. De twee Logo-opdrachten zijn:

TOOT en SETENV

TOOT is de opdracht die een geluidskanaal opent. In Atari Logo zijn (zonder het gebruik van machinetaal) twee geluidskanalen beschikbaar. In BASIC zijn er maar liefst vier geluidskanalen

'direct' te gebruiken. Deze geluidskanalen roepen we met respectievelijk een 0 of een 1 aan.

De opdracht TOOT vraagt om vier invoergegevens. U weet het, toetsen we alleen TOOT in en geven we daarna RETURN, dan zegt Logo ons dat de boodschap niet begrepen wordt:

```
NOT ENOUGH INPUTS TO TOOT
```

Dus moeten we iets aan TOOT toevoegen. En wel het volgende:

```
TOOT :stem :frequentie :volume :lengte
```

- Het eerste invoergegeven is de stem, het geluidskanaal. We kunnen hier kiezen tussen het eerste (0) of het tweede (1) geluidskanaal.
- Het tweede invoergegeven is de frequentie, de hoogte van de toon. In dit hoofdstuk vindt u een frequentietabel. De toon A heeft bijvoorbeeld een frequentie van 440 Hz.
- Het derde invoergegeven is het volume van de toon (maximaal 15). Hoe hoger het getal, hoe groter het volume van de toon.
- Het vierde invoergegeven tenslotte bepaalt de duur van de toon. Deze tijdsduur (een getal tussen 0 en 255) wordt aangegeven in eenheden van 1/60 seconde. Geven we een toon dus een duur van 90, dan zal deze toon anderhalve seconde duren.

Zo geeft

```
TOOT 1 440 10 30
```

een toon uit het tweede kanaal, met een hoogte van 440 Hz, een volume van 10 en een duur van een halve seconde. Een A dus.

Wat zou u denken van de procedure A?

```
EDIT "A
TOOT 0 440 10 30
END
```

Nu is voor het laten horen van de toon A het aanroepen van de procedure, die ook A heet, voldoende.

Op dezelfde manier als de procedure A gemaakt is zouden we de hele toonladder kunnen afwerken:

C D E F G A B C

Maar nu zien we dat er tweemaal een toon met dezelfde naam voorkomt. Als u meer dan één oktaaf definieert, krijgt u nog meer dubbele namen. En u weet dat het in Logo onmogelijk is om twee procedures met dezelfde naam tegelijk in het geheugen te hebben. Daarom zullen we de ene procedure bijvoorbeeld C en de andere C2 moeten noemen.

Ook is het verstandig om in de procedures variabelen te hante-

ren. Een voorbeeld:

```
EDIT "C3 :DUUR
TOOT 0 1046 :VOLUME :DUUR * 15
END
```

Zoals u ziet is in deze procedure zowel het volume als de duur variabel. Omdat het in de meeste muziekstukjes niet zinvol is om voor iedere toon steeds opnieuw het volume te bepalen, kunnen we de variabele :VOLUME aan het begin van het muziekstuk zijn waarde geven. Wanneer we in het muziekstuk een 'zacht' gedeelte krijgen, kunnen we de waarde in de procedure nogmaals bijstellen. Voorbeelden hiervan vindt u verderop in dit hoofdstuk.

Let op. Omdat de waarde van :DUUR in de procedure met 15 vermenigvuldigd wordt, duurt C3 4 een volle seconde.

Het schrijven van een procedure voor iedere toon is echter niet de enige manier om muziek te maken. Het is ook mogelijk om bepaalde gegevens: de frequentie, het volume en/of de duur, in een lijst te zetten. Een voorbeeld:

```
MAKE "GEGEVENS [349 349 523 523 587 587 523]

EDIT "KORTJAKJE :GEGEVENS
IF EMPTYP :GEGEVENS [STOP]
MAKE "TOON FIRST :GEGEVENS
TOOT 0 :TOON 10 30
KORTJAKJE BUTFIRST :GEGEVENS
END
```

Dit is een heel kort en heel eenvoudig voorbeeld, maar het gaat om het principe.

Zoals gezegd kunnen we ook bijvoorbeeld de duur van de noten in een gegevenslijst zetten. U kunt dat op twee manieren doen:

- twee aparte lijsten maken: een met de frequentiegegevens en een met de gegevens voor de toonduur:

```
MAKE "HOOGTE [349 523 587 523]
MAKE "DUUR [30 60 30 15]
```

Een procedure kan er dan zo uitzien:

```
EDIT "LIED :HOOGTE :DUUR
IF OR EMPTYP :HOOGTE EMPTYP :DUUR [STOP]
MAKE "H FIRST :HOOGTE
MAKE "D FIRST :DUUR
TOOT 0 :H 15 :D
LIED BF :H BF :D
END
```

- één lijst maken, waarbij ieder element twee gegevens bevat (hoogte en duur):

```
MAKE "GEGEVENS [[349 30][523 60][587 30][523 15]]
```

De procedure ziet er dan zo uit:

```
EDIT "LIED :GEGEVENS
IF EMPTYP :GEGEVENS [STOP]
MAKE "H FIRST FIRST :GEGEVENS
MAKE "D LAST FIRST :GEGEVENS
TOOT 0 :H 15 :D
LIED BF :GEGEVENS
END
```

U ziet het, mogelijkheden te over.

U kunt zelf bepalen welke methode u hanteert. Beide methoden (voor iedere toon een procedure maken of de gegevens voor de tonen in een lijst zetten) hebben hun voordelen.

Hieronder vindt u de tabel met de frequentiewaarden van de verschillende tonen.

Opmerking. Er zijn minieme verschillen in frequentiewaarden mogelijk. Ga dit na voor uw eigen computer.

Tabel met frequentiewaarden van de verschillende tonen

C = 261 Hz	G = 391 Hz	Cis2 = 554 Hz	G2 = 783 Hz
Cis = 277 Hz	Gis = 415 Hz	D2 = 587 Hz	Gis2 = 830 Hz
D = 293 Hz	A = 440 Hz	Dis2 = 622 Hz	A2 = 880 Hz
Dis = 311 Hz	Ais = 466 Hz	E2 = 659 Hz	Ais2 = 932 Hz
E = 329 Hz	B = 493 Hz	F2 = 698 Hz	B2 = 987 Hz
F = 349 Hz	C2 = 523 Hz	Fis2 = 739 Hz	C3 = 1046 Hz
Fis = 369 Hz			

Laten we maar eens een muziekstukje maken om een en ander in praktijk te brengen. We gaan het bekende liedje *Altijd is Kortjakje ziek* op twee manieren programmeren. Kijkt u maar:

1. Iedere noot heeft een eigen procedure

We hebben de volgende noten nodig: F G A Ais C2 D2. Iedere procedure moet twee variabelen bevatten: de toonduur en het volume. De procedure D2 ziet er dan zo uit:

```
EDIT "D2 :DUUR
TOOT 0 578 :VOLUME :DUUR * 15
END
```

En nu het lied:

```
EDIT "KORTJAKJE
MAKE "VOLUME 15
F 2 F 2 C2 2 C2 2 D2 2 D2 2 C2 4
Ais 1 Ais 1 Ais 1 Ais 1 A 2 A 2 G 2 G 2 F 4
MAKE "VOLUME 10
REPEAT 2 [C2 2 C2 2 Ais 2 Ais 2 A 2 A 2 G 4]
MAKE "VOLUME 15
F 2 F 2 C2 2 C2 2 D2 2 D2 2 C2 4
Ais 1 Ais 1 Ais 1 Ais 1 A 2 A 2 G 2 G 2 F 4
END
```

U ziet het: de waarde van :VOLUME wordt gedurende het muziekstuk gewijzigd. Dit kan zonder problemen. Het is een van de voordelen van het op deze manier muziek maken.

Wanneer we de benodigde 'toon'-procedures eenmaal hebben, kunnen we ook zeer snel een volgend muziekstuk maken.

Natuurlijk zitten er ook nadelen aan deze manier van muziekprocedures maken. Het is eigenlijk een nogal omslachtige manier. Laten we eens naar methode 2 kijken.

2. Bij deze methode gebruiken we maar één procedure, namelijk

```
EDIT "LIED :GEGEVENS
IF EMPTYP :GEGEVENS [STOP]
MAKE "HOOGTE FIRST FIRST :GEGEVENS
MAKE "VOLUME FIRST BF FIRST :GEGEVENS
MAKE "DUUR LAST FIRST :GEGEVENS
TOOT 0 :HOOGTE :VOLUME :DUUR
LIED BF :GEGEVENS
END
```

Met deze procedure kunnen we ieder gewenst eenstemmig muziekstuk maken. Het enige wat we moeten doen is steeds de gegevenslijst vullen met nieuwe informatie.

De gegevenslijst bevat zoveel elementen als er noten zijn. Ieder element bevat dus de informatie over één noot. Deze informatie betreft de hoogte, het volume en de duur van de noot. Ieder element van de gegevenslijst bevat dus drie elementen.

Deze elementen worden in LIED als volgt benoemd:

:HOOGTE = het eerste element van het eerste element
(FIRST FIRST)
:VOLUME = het middelste element van het eerste element
(FIRST BF FIRST)
:DUUR = het laatste element van het eerste element
(LAST FIRST)

Het eerste element ziet er dan zo uit:

```
[349 15 30]
```

U ziet, de hoogte is 349 Hz (een F), het volume 15 en de duur 30.

Dit is dan de volledige gegevenslijst voor Kortjakje:

```
MAKE "GEGEVENS [[349 15 30][349 15 30][523 15 30]
[523 15 30][587 15 30][587 15 30][523 15 60][466 15 15]
[466 15 15][466 15 15][466 15 15][440 15 30][440 15 30]
[391 15 30][391 15 30][349 15 60][523 10 30][523 10 30]
[466 10 30][466 10 30][440 10 30][440 10 30][391 10 60]
[523 10 30][523 10 30][466 10 30][466 10 30][440 10 30]
[440 10 30][391 10 60][349 15 30][349 15 30][523 15 30]
[523 15 30][587 15 30][587 15 30][523 15 60][466 15 15]
[466 15 15][466 15 15][466 15 15][440 15 30][440 15 30]
[391 15 30][391 15 30][349 15 60]]
```

Wanneer we deze gegevens invoeren in de procedure LIED, is het resultaat het liedje *Altijd is Kortjakje ziek*.

En als we een ander muziekstukje willen maken? Dat is geen enkel probleem. We maken gewoon een nieuwe gegevenslijst met nieuwe informatie. We kunnen de procedure LIED blijven gebruiken. Dit is een zeer handige methode om met Logo muziek te maken.

5.4 Meerstemmige muziek met Logo

Zoals al eerder is vermeld, beschikt Atari LCSI Logo over twee geluidskanalen die ieder apart, maar ook tegelijk aangeroepen kunnen worden. Tot nu toe hebben we ons steeds beperkt tot het eerste geluidskanaal. Dat riepen we aan met 0. Nu komt het tweede geluidskanaal (register zo u wilt) erbij. Dat kanaal roepen we aan met het cijfer 1.

Indien u met losse 'toonprocedures' wilt werken, dus voor iedere toon een eigen procedure wilt maken, zult u ook voor de tonen uit het tweede register eigen procedures moeten maken. De 'toonprocedures voor het eerste kanaal waren:

C Cis D Dis E F Fis G Gis A Ais B C2 Cis2 D2 Dis2 E2 F2 Fis2
G2 Gis2 A2 Ais2 B2 C3

De 'toonprocedures' voor kanaal 1 zou u dan als volgt kunnen benoemen:

c cis d dis e f fis g gis a ais b c2 cis2 d2 dis2 e2 f2 fis2
g2 gis2 a2 ais2 b2 c3

Een procedure voor het eerste kanaal krijgt (of had reeds) de opdracht

TOOT 0

en een procedure voor het tweede kanaal krijgt de opdracht

TOOT 1

Nu zullen er dus 25 nieuwe procedures geschreven moeten worden. Deze nieuwe procedures zijn echter bijna gelijk aan de reeds gemaakte toonprocedures voor kanaal 0. U hoeft alle procedures dus niet weer opnieuw te schrijven. Er is een handiger manier, waarbij u de reeds geschreven procedures gebruikt, kijkt u maar:

1. Roep eerst de procedure C in de Editor met EDIT "C.
2. Verander in de Editor de naam van de procedure (van C naar c) en het geluidskanaal (van 0 naar 1).
3. Verlaat de Editor met de ESC-toets (voor Atari).
4. We hebben nu zowel de procedure C als de procedure c.
5. Doe dit met alle toonprocedures.

Als we nu tweestemmig muziek willen maken moeten we de geluidskanalen om de beurt aanroepen. Dat gaat op deze wijze:

C 2 c 2 C 2 c 2 E 2 e 2 F 2 f 2 C 4 c 4

Als u goed luistert hoort u wel twee geluidskanalen (het geluid is wat voller), maar de 'muziek' is nog niet tweestemmig. Het geluid wordt pas tweestemmig als we twee verschillende noten tegelijk laten spelen, als een harmonisch interval:

C 2 e 2 F 2 ais 2 D 2 a 2 G 2 d 2

Het resultaat hiervan is een tweestemmige regel.

Dit is één mogelijkheid om tweestemmig te werken; er is echter nog een andere manier. Ook hier kunnen we namelijk de informatie weer in een lijst plaatsen.

Willen we de gegevens voor tweestemmige Logo-muziek in een lijst zetten, dan zijn er verschillende mogelijkheden:

1. We maken één lijst waarbij ieder element zes gegevens bevat:
   - de toonhoogte van de toon uit kanaal 0
   - het volume van de toon uit kanaal 0
   - de duur van de toon uit kanaal 0
   - de toonhoogte van de toon uit kanaal 1
   - het volume van de toon uit kanaal 1
   - de duur van de toon uit kanaal 1

Als u op deze manier werkt bent u echter snel 'de draad kwijt', want het geheel is niet erg overzichtelijk.

2. We maken twee lijsten, één met alle gegevens voor kanaal 0 en één met alle gegevens voor kanaal 1.

Deze lijsten zien er dan zo uit:

```
MAKE "GEGEVENS.0 [[261 15 30][329 15 30][391 15 30]
                 [261 15 30]]
MAKE "GEGEVENS.1 [[440 10 30][440 10 30][493 10 30]
                 [523 10 30]]
```

En ook de procedure ziet er een beetje anders uit dan voor eenstemmige muziek:

```
EDIT "LIED2 :GEGEVENS.0 :GEGEVENS.1
IF OR EMPTYP :GEGEVENS.0 EMPTYP :GEGEVENS.1 [STOP]
MAKE "H.0 FIRST FIRST :GEGEVENS.0
MAKE "V.0 FIRST BF FIRST :GEGEVENS.0
MAKE "D.0 LAST FIRST :GEGEVENS.0
MAKE "H.1 FIRST FIRST :GEGEVENS.1
MAKE "V.1 FIRST BF FIRST :GEGEVENS.1
MAKE "D.1 LAST FIRST :GEGEVENS.1
TOOT 0 :H.0 :V.0 :D.0
TOOT 1 :H.1 :V.1 :D.1
LIED2 BF :GEGEVENS.0 BF :GEGEVENS.1
END
```

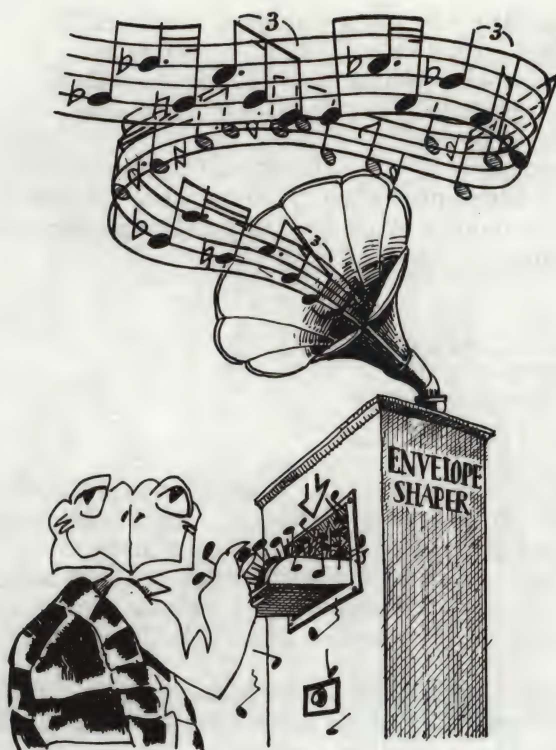
Er zijn natuurlijk nog meer manieren om de gegevens in een lijst vast te leggen en deze er door de procedure LIED2 weer uit te laten halen. U kunt bijvoorbeeld een gegevenslijst maken waarbij het eerste element de gegevens bevat voor de eerste toon uit kanaal 0, het tweede element de gegevens voor de eerste toon uit kanaal 1, enzovoort. Maar daar gaan we hier niet verder op in.

Met behulp van de zojuist beschreven werkwijze en met gebruikmaking van de procedures uit dit hoofdstuk kunt u naar hartelust experimenteren met Logo-muziek.

We zijn er echter nog niet. We hebben het nog slechts over één muziekopdracht gehad. Het Logo-woord SETENV hebben we nog helemaal niet besproken.

5.5 Het klankverloop beheersen met de 'Envelope Shaper'

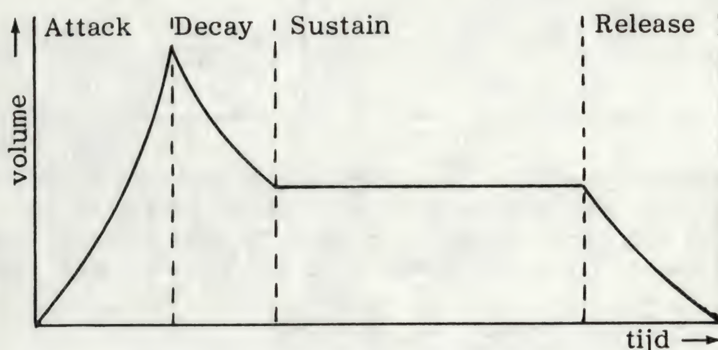
Met de opdracht SETENV kunnen we het klankverloop (de envelope) van een toon beheersen en controleren. SETENV is de afkorting van SET ENVelope.



Natuurlijk kunnen we met de opdracht TOOT alleen ook wel muziek maken. Voor iedere toon kunnen we met TOOT het geluidskanaal, de hoogte, het volume en de duur bepalen. Toch klinkt alles dan nog wat monotoon. Iedere toon blijft (zolang hij duurt) steeds op één volumeniveau. Er is geen verloop, geen 'envelope'.

Met de 'Envelope Shaper' en de opdracht SETENV kunnen we hier iets aan veranderen.

Weet u nog hoe het verloop van een toon eruit zag?



Meteen na de aanslag een hoog volume	: Attack
daarna iets teruglopend	: Decay
tot een min of meer constant niveau	: Sustain
en tenslotte uitstervend	: Release

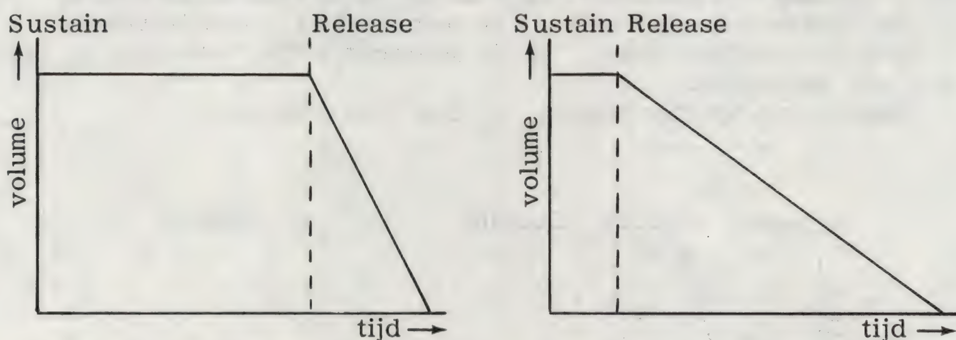
Opmerking. We zullen in deze paragraaf verder de Engelse termen gebruiken. Bij Nederlandstalige Logo-versies zijn deze termen veelal verschillend vertaald. Malmberg Logo (Commodore) gebruikt de volgende vertaling:

Attack	: Het aanzwellen
Decay	: Het vervallen
Sustain	: Het nagalman
Release	: Het uitsterven

Het zou prettig zijn als we ook in Logo de mogelijkheden hadden om het klankverloop zo uitvoerig te beheersen. Sommige Logo-versies hebben inderdaad geweldige mogelijkheden hiervoor.

De meeste Logo-versies echter moeten het met minder mogelijkheden doen, en kunnen daarom de Attack, de Decay, de Sustain en de Release niet apart instellen. SETENV is echter een redelijk alternatief.

In Atari LCSI Logo zijn de Attack en de Decay niet door SETENV te beïnvloeden. Met de opdracht SETENV kunnen we slechts de Sustain en de Release regelen. Om precies te zijn: met SETENV bepalen we het punt waar de Sustain overgaat in de Release. We kunnen ook zeggen: met SETENV bepalen we hoe lang een toon, op vol volume, hoorbaar moet zijn en hoe lang de uitsterftijd van een toon is. Dit heeft niets met de duur van een toon te maken. Kijk maar eens naar de volgende twee grafieken:



Laten we aannemen dat dit de grafieken zijn van de toon A. De linkergrafiek toont een A met een lange Sustain, terwijl de rechtergrafiek juist een korte Sustain laat zien. De twee tonen klinken dan ook verschillend, hoewel ze dezelfde specificaties kunnen hebben:

hoogte: 440 Hz volume: 15 duur: 30

De eerste grafiek echter laat een A zien die vrij lang hoorbaar is en dan vrij plotseling uitsterft. De toon is zeker 18 van de 30 eenheden hoorbaar. Daarna zakt de toon in zeer korte tijd snel in volume omlaag. We spreken hier van een lange Sustain en een korte Release.

De tweede grafiek laat eveneens een A zien, met dit verschil dat deze toon slechts kort hoorbaar is. De Sustain van deze toon is erg kort. De Release, de uitsterftijd, echter is erg lang.

In muziektermen kunnen we zeggen, dat tonen met een lange Sustain 'legato' klinken (dat is: verbonden) en tonen met een korte Sustain 'staccato'.

Het moment nu waarop de Sustain overgaat in de Release kunnen we met SETENV regelen. We zullen dit moment (waarop de Sustain overgaat in de Release) s/r.punt noemen. De opdracht SETENV gebruiken we zo:

```
SETENV :geluidskanaal :s/r.punt
```

Het eerste invoergegeven is het geluidskanaal waarvoor het klankverloop wordt bepaald. Hier kan dus een 0 of een 1 worden ingevuld. Het tweede invoergegeven is het moment waarop de Sustain moet overgaan in de Release: het s/r.punt. Dit kan een getal zijn tussen 0 en 127.

Het beste kunnen we een en ander illustreren met een muziekvoorbeeld. Het is een betrekkelijk eenvoudige procedure die u een muziekstukje laat horen en daarbij aangeeft wat de SETENV-waarde is. Deze SETENV-waarde verandert telkens als het muziekstukje opnieuw begint. Zo kunt u het duidelijke verschil horen tussen SETENV 0 125 en SETENV 0 5. Geef om de procedure te starten een invoergegeven: de beginwaarde van de 'Envelope Shaper'.

```
MAKE "GEGEVENS [[523 10 10][698 10 10][659 10 10]
[698 10 10][659 10 5][587 10 5][523 10 5][466 10 5]
[440 10 5][466 10 5][440 10 10][391 10 10][349 10 20]]

EDIT "TEST :INVOER
IF :INVOER = "0 [STOP]
PR (SE [De SETENV waarde is :] :INVOER
SETENV 0 :INVOER
LIED :GEGEVENS
MAKE "INVOER :INVOER - 5
TEST :INVOER
END
```

Let op: als verkleiningsfactor voor de waarde van de Envelope Shaper is 5 gekozen. U kunt dit vanzelfsprekend aanpassen aan uw eigen wensen. Begin de procedure dus met een 'hoge' SETENV-waarde. De procedure "TEST zal dan het in :GEGEVENS opgeslagen melodietje blijven spelen en iedere keer de waarde van de Envelope Shaper verminderen.

Omdat TEST recursief is gaat dit proces door totdat de waarde van de variabele :INVOER gelijk is aan 0. Door nu de Envelope Shaper als invoer een 0 te geven maken we hem inactief.

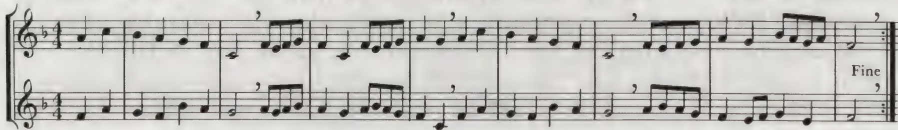
Twee meerstemmige muziekvoorbeelden

Laten we eens gaan kijken naar enkele voorbeelden van meerstemmige muziek. Het eerste is een voorbeeld van een muziekprocedure die gebruik maakt van afzonderlijke 'toonprocedures'. Het tweede muziekstuk is een voorbeeld van een procedure met een gegevenslijst.

1. *Vous quidonnez de l'amour (1730)*

Dit is het eerste gedeelte van een oud Frans volkswijsje. Let hierbij vooral eens op de steeds veranderende SETENV-waarde. Een lage SETENV-waarde wordt gebruikt om noten, op gelijke hoogte, die elkaar opvolgen ook als afzonderlijke noten te laten klinken. Een hoge SETENV-waarde wordt gebruikt om noten, op gelijke hoogte, die elkaar opvolgen 'verbonden' te laten klinken. Bij een zeer hoge SETENV-waarde klinken twee korte (gelijke) noten zelfs als één lange noot!

Hier volgen de muziek en de procedure.




```
EDIT "L.AMOUR
MAKE "volume 10
SETENV 0 100 SETENV 1 100
A 2 f 2 C2 2 a 2 Ais 2 g 2 A 2
f 2 G 2 ais 2 F 2 a 2 C 4 g 4
SETENV 0 5
F 1 a 1 E 1 g 1 F 1 a 1 G 1
ais 1
SETENV 0 100
F 2 a 2 C 2 g 2
SETENV 0 5 SETENV 1 5
F 1 a 1 E 1 ais 1 F 1 a 1 G 1
g 1
SETENV 0 100 SETENV 1 100
A 2 f 2 G 2 c 2 A 2 f 2 C2 2
a 2 Ais 2 g 2 A 2 f 2 G 2
ais 2 F 2 a 2 C 4 g 4
```

→

```
SETENV 0 0 SETENV 1 2
F 1 a 1 E 1 ais 1 F 1 a 1 G 1
g 1 A 2 f 2 G 1 e 1 G 1 f 1
SETENV 0 2 SETENV 1 0
Ais 1 g 1 A 1 g 1 G 1 e 1 A 1
e 1 F 4 f 4
END
```

Opvallend in deze muziekprocedure is het feit dat een kwart-noot als twee achtste-noten genoteerd wordt. Een notatie als:

G 2 e 1 f 1
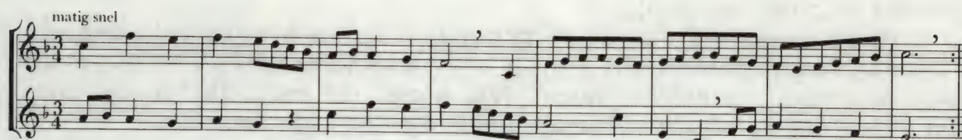
is in dit geval namelijk niet mogelijk. De tonen zullen dan om de beurt en niet tegelijk worden gespeeld. Daarom wordt een en ander als volgt genoteerd:

G 1 e 1 G 1 f 1

U ziet, de lange G wordt als het ware in tweeën gespeeld, maar door nu de SETENV hoog te kiezen (of op 0 te zetten) hoort u geen twee aparte noten.

2. *Het Menuet 'La Marguerite'*

'La Marguerite is een Franse dans uit 1730 en werd oorspronkelijk geschreven voor doedelzak en viool. Dit is de muziek:



En dit zijn de gegevens:

```
MAKE "GEGEVENS.0 [[523 10 10][523 10 10][698 10 20]
[659 10 20][698 10 20][659 10 10][587 10 10][523 10 10]
[466 10 10][440 10 10][466 10 10][440 10 20][391 10 20]
[349 10 10][349 10 10][349 10 10][349 10 10][261 10 10]
[261 10 10][349 15 10][391 15 10][440 15 10][440 14 10]
[391 14 10][349 13 10][391 15 10][440 15 10][466 15 10]
[466 14 10][440 14 10][391 13 10][349 15 10][329 15 10]
[349 15 10][391 14 10][440 14 10][466 13 10][523 15 60]]
```

```
MAKE "GEGEVENS.1 [[440 10 10][466 10 10][440 10 20]
[391 10 20][440 10 20][391 10 10][391 10 10][50000 2 10]
[50000 2 10][523 10 10][523 10 10] [698 10 20]
[659 10 20][698 10 10][698 10 10][659 10 10][587 10 10]
[523 10 10][466 10 10][440 15 10][440 14 10][440 13 10]
[440 12 10][523 11 10][523 10 10][329 15 10][329 14 10]
[261 13 10][261 12 10][349 11 10][391 10 10][440 15 10]
[440 14 10][391 13 10][391 12 10][349 11 10][349 10 10]
[329 15 60]]
```

Dit is de gebruikte procedure:

```
EDIT "MARGUERITE :GEGEVENS.0 :GEGEVENS.1
IF OR EMPTYP :GEGEVENS.0 EMPTYP :GEGEVENS.1 [STOP]
MAKE "H.0 FIRST FIRST :GEGEVENS.0
MAKE "V.0 FIRST BF FIRST :GEGEVENS.0
MAKE "D.0 LAST FIRST :GEGEVENS.0
MAKE "H.1 FIRST FIRST :GEGEVENS.1
MAKE "V.1 FIRST BF FIRST :GEGEVENS.1
MAKE "D.1 LAST FIRST :GEGEVENS.1
SETENV 0 3 SETENV 1 100
TOOT 0 :H.0 :V.0 :D.0
TOOT 1 :H.1 :V.1 :D.1
MARGUERITE BF :GEGEVENS.0 BF :GEGEVENS.1
END
```

U ziet het, in een muziekprocedure als deze wordt de opdracht SETENV weinig toegepast. De SETENV-waarde blijft gedurende het gehele muziekstuk gelijk.

In de gegevens van de afzonderlijke noten worden echter de accenten aangegeven. Dit zijn de accenten voor de eerste noten in een maat (aangegeven door een iets hoger volume). Dit volume loopt langzaam terug bij de volgende noten in dezelfde maat.

5.6 *Tot besluit: Turtle Music*

Ter afsluiting van dit hoofdstuk over het muziek-maken met Logo nog het programma Turtle Music. Het is een betrekkelijk eenvoudig programma om muziek mee te maken. Op het scherm wordt dan ook een pianoklavier getekend. Dit gebeurt met de volgende procedures:

```
TO V2
LT 90 FD 10 RT 90
END

TO ZWARTE.TOETS
REPEAT 5 [FD 30 SETX XCOR + 1 BK 30] LT 90 FD 5 RT
 90
END

TO TOETS
REPEAT 2 [FD 50 RT 90 FD 10 RT 90]
END

TO KLAVIER
SETBG 7 CS CT SS
SETPC 0 0
HT PU SETPOS [-153 -10]
PD REPEAT 31 [TOETS RT 90 FD 10 LT 90]
PU SETPOS [-146 40] RT 180 PD
REPEAT 4 [REPEAT 2 [ZWARTE.TOETS] V2 REPEAT 3 [ZWA
RTE.TOETS] V2]
REPEAT 2 [ZWARTE.TOETS]
PU
LEER
END
```

Hierna wordt de procedure LEER aangeroepen. Deze procedure vraagt om een naam voor de te spelen 'melodie' en roept dan op zijn beurt de procedure NEEMOP aan.

NEEMOP beschouwt iedere gespeelde noot als een element waarmee de lijst :melodie wordt gevuld. :melodie wordt dus de variabele die alle noten van het gespeelde bevat.

De noten worden via het toetsenbord gespeeld met kleine letters. Voor de verhoogde noten worden HOOFDLETTERS gebruikt.
Bijvoorbeeld: de noot f wordt gespeeld als u intoetst: f;
de noot f# wordt gespeeld als u intoetst: F (SHIFT f).

Met ZET wordt de Turtle op de juiste toets van het klavier geplaatst. ZW maakt de Turtle zwart (voor op de witte toetsen). WI maakt de Turtle wit (voor op de zwarte toetsen).

Dit zijn de procedures:

```
TO LEER
TELL 0 ST SETPOS [-9 0] SETH 0
CT SS PR [Hoe komt uw melodie te heten ?]
MAKE "naam RW
MAKE "melodie []
NEEMOP
MAKE THING "naam :melodie
END
```

```
TO NEEMOP
MAKE "T RC
IF :T = CHAR 32 [STOP]
MAKE "V 10 MAKE "L 10
SETENV 0 0
NOTEN :T
ZET :T
MAKE "melodie LPUT :T :melodie
NEEMOP
END

TO NOTEN :T
IF :T = CHAR 99 [TOOT 0 261 :V :L]
IF :T = CHAR 67 [TOOT 0 277 :V :L]
IF :T = CHAR 100 [TOOT 0 293 :V :L]
IF :T = CHAR 68 [TOOT 0 311 :V :L]
IF :T = CHAR 101 [TOOT 0 329 :V :L]
IF :T = CHAR 102 [TOOT 0 349 :V :L]
IF :T = CHAR 70 [TOOT 0 369 :V :L]
IF :T = CHAR 103 [TOOT 0 391 :V :L]
IF :T = CHAR 71 [TOOT 0 415 :V :L]
IF :T = CHAR 97 [TOOT 0 440 :V :L]
IF :T = CHAR 65 [TOOT 0 466 :V :L]
IF :T = CHAR 98 [TOOT 0 493 :V :L]
IF :T = CHAR 27 [TOOT 0 523 :V :L]
END

TO ZET :T
IF :T = CHAR 99 [SETPOS [-9 0] ZW]
IF :T = CHAR 67 [SETPOS [-4 15] WI]
IF :T = CHAR 100 [SETPOS [1 0] ZW]
IF :T = CHAR 68 [SETPOS [6 15] WI]
IF :T = CHAR 101 [SETPOS [11 0] ZW]
IF :T = CHAR 102 [SETPOS [21 0] ZW]
IF :T = CHAR 70 [SETPOS [26 15] WI]
IF :T = CHAR 103 [SETPOS [31 0] ZW]
IF :T = CHAR 71 [SETPOS [36 15] WI]
IF :T = CHAR 97 [SETPOS [41 0] ZW]
IF :T = CHAR 65 [SETPOS [46 15] WI]
IF :T = CHAR 98 [SETPOS [51 0] ZW]
IF :T = CHAR 27 [SETPOS [61 0] ZW]
END

TO RW
OP FIRST RL
END
```

```
TO WI
SETC 7
END
```

```
TO ZW
SETC 0
END
```

Zoals u waarschijnlijk reeds had ontdekt wordt het 'opnemen' beëindigd met de spatietoets (= CHAR 32).

Na het 'opnemen' van een melodie kan deze vanzelfsprekend ook weer beluisterd worden. Dat gebeurt door de procedure SPEEL. Toets daarvoor, na het opnemen, SPEEL :naam in. Vanzelfsprekend dient u hierbij de naam op te geven van de melodie die gespeeld moet worden.

Op deze manier zijn vele melodieën op te nemen en af te spelen.

Denkt u eraan voor de naam van de melodie een : te plaatsen? Anders antwoordt Logo:

```
I DON'T KNOW HOW TO naam
```

Een dubbele punt wil zeggen dat :naam geen procedure maar een variabele is.

Dit is dan de procedure SPEEL:

```
TO SPEEL :melodie
IF EMPTYP :melodie [STOP]
MAKE "V 10 MAKE "L 10
SETENV 0 0
MAKE "T FIRST :melodie
NOTEN :T
ZET :T
SPEEL BF :melodie
END
```

Zoals al opgemerkt is dit een betrekkelijk eenvoudig programma met niet zo heel veel mogelijkheden. Het volume, de toonduur en de instelling van de Envelope Shaper zijn niet variabel maar liggen vast. Met dit programma als uitgangspunt kunt u er echter wel iets van maken. Logo biedt u hier mogelijkheden genoeg voor.
Veel plezier!

Hoofdstuk 6

DE WERELD VAN DE LOGO-SPROKEN



6.1 Sproken, wat zijn dat?

Wie heeft er wel eens van sproken gehoord? Spoken ... ja; sprookjes ook ... maar sproken? Veel computer-enthousiasten zullen u niet kunnen vertellen wat sproken zijn.

Zijn sproken dan zo exclusief voor Logo? Niet echt, want ook mensen die in BASIC of machinetaal programmeren kennen ze. Zegt de term 'Player Missile Graphics' of 'Sprites' u meer? Dat zijn die kleine figuurtjes op het beeldscherm: rondrijdende autootjes, rondrennende mannetjes/vrouwtjes, vliegtuigjes, helicoptertjes, raketjes, ...

Heeft u wel eens geprobeerd Sprites te maken vanuit BASIC? Iedereen die deze vraag bevestigend kan beantwoorden weet dat het een hele klus is. Daar komt nog bij, dat Sprites, in BASIC geprogrammeerd, vaak - niet altijd - traag zijn in het bewegen en schokkerig reageren. Dit is niet het geval bij in machinetaal geprogrammeerde Sprites. Iedereen kent ze wel: de razendsnelle computerspelen in de meest fantastische kleuren.

Welnu, Sprites kunnen we ook met Logo maken. En wel op een zeer gebruikersvriendelijke manier.

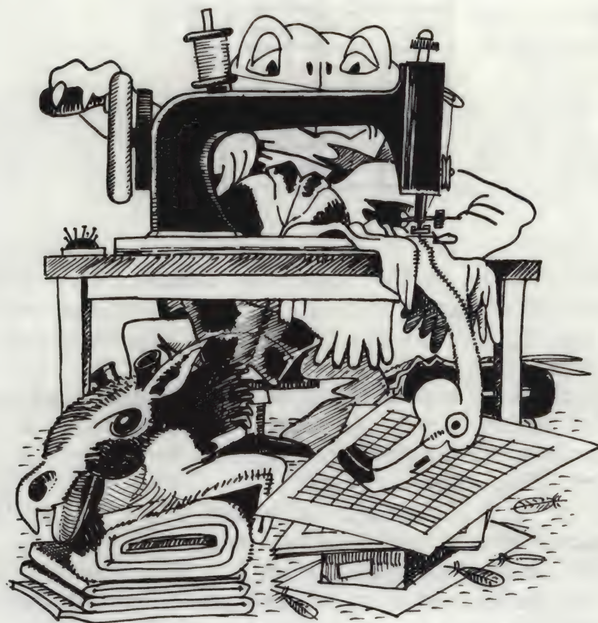
Deze Logo Sprites, die we voortaan gewoon sproken zullen noemen, hebben enkele belangrijke kenmerken:

- Er kan met meerdere sproken tegelijk (en onafhankelijk van elkaar) worden gewerkt. Atari LCSI Logo beschikt over vier sproken, BBC en Commodore over meer.
- Sproken zijn eenvoudig te ontwerpen.
- Atari LCSI sproken zijn gedetailleerd: geen 8 maar 16 bytes.
- Sproken zijn eenvoudig te bewegen: via toetsenbord, joystick of paddle.
- Sproken kunnen met POKE-waarden groter of kleiner worden gemaakt.
- Sproken kunnen vele kleuren aannemen (bij Atari 128!).
- Sproken kunnen alle vormen aannemen, uw eigen fantasie of 128 beeldpunten is de grens.
- Er is een uitgebreide lijst voor het opsporen en vermijden van botsingen. In Logo wordt dit 'Collision Detection' genoemd. Maar daarover later meer.

Eigenlijk is een Logo-sprook niets anders dan de Logo-schildpad in een ander jasje. Willen we de Logo-schildpad 'verkleden' als auto, paard, rots, vogel of vrachtwagen? Dat is geen enkel probleem. Met Logo-sproken is iedereen in staat zijn/haar eigen computerspelen te maken. Vooral wanneer Logo-sproken worden gecombineerd met Turtle-Graphics, met Logo-taal en/of met Logo-muziek zijn er bijzonder aardige dingen mee te doen.

Zullen we eens gaan kijken hoe een sprook wordt gemaakt?

6.2 Het maken en ontwerpen van sproken



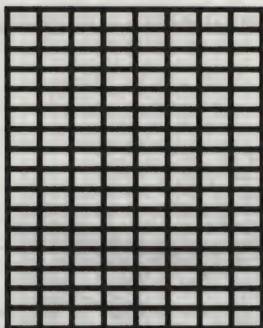
Bij Atari LCSI Logo kunnen maximaal vijftien sproken (door vier Turtles) tegelijk worden gebruikt. De zestiende sprook is de Turtle zelf.

Sproken maken en ontwerpen doen we in een raster van 128 hokjes. Dit raster, dat de 'Shape Editor' wordt genoemd, is uw ontwerpblad. Door het inkleuren/openlaten van de hokjes in het raster maakt u een sprook.

Opmerking. Ook hier zal de Engelse naam worden gebruikt. Shape Editor is een Logo-begrip dat moeilijk te vertalen is.

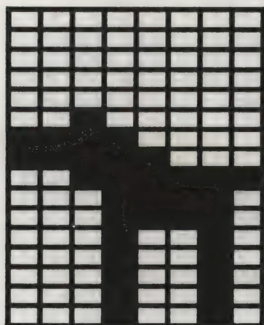
De Shape Editor roepen we op met de opdracht: EDSH (= Edit Shape). EDSH moet gevolgd worden door een getal (1-15).

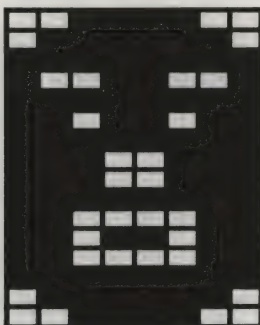
We willen nu de eerste sprook ontwerpen, dus toetsen we EDSH 1 in (en RETURN). Dan verschijnt dit op het beeldscherm:

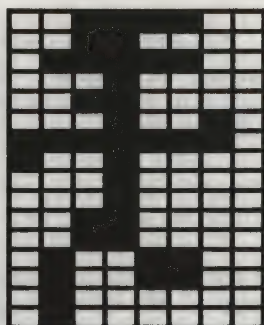


We zijn nu in de Shape Editor. U ziet het raster (8 x 16 = 128 hokjes), uw 'tekenvel'. U kunt uw eigen sprook ontwerpen. Het ontwerp mag wat grof lijken, het uiteindelijke resultaat, de sprook, is vele malen kleiner.
Hoe kan een sprook er nu uitzien? Hieronder ziet u enkele figuren uit een spel dat we ooit eens gemaakt hebben. Het spel heette: Het Grote Yama Avontuur, en de drie afgebeelde figuren speelden hierin een grote rol. Het waren: Pegus, het paard; het tekenduiveltje en Luut, de Loper. U kunt zich wel voorstellen dat er heel veel verschillende figuren ontworpen kunnen worden.

Later in dit hoofdstuk zult u zien hoe u ook letters en cijfers kunt maken. Ja, zelfs hoe u de letters en cijfers uit het ROM-geheugen van uw computer kunt halen en in de Shape Editor kunt stoppen.







Het lijkt misschien een nadeel dat sproken in een rechthoekig raster ontworpen moeten worden. Zo zou u alleen rechtopstaande figuren kunnen maken. Maar ... ook in Logo hebt u de beschikking over

de POKE-waarden van uw computer. Met behulp van de juiste POKE-waarde maakt u de sprook breder.

Opmerking. De opdrachten breder, smaller, groter en kleiner zitten standaard in een aantal Logo's. Het werken met POKE-waarden is hierbij niet nodig.

De opdracht om een waarde te 'poken' is .DEPOSIT. Deze opdracht vraagt om twee invoergegevens:
- de geheugenplaats die wordt aangeroepen
- de waarde die in de geheugenplaats wordt gezet.

Denkt u om de stip?

U weet dat iedere computer beschikt over andere geheugenplaatsen. Als voorbeeld volgen hier de .DEPOSIT-waarden voor de Atari:

.DEPOSIT 53256 1 (of 3) maakt Turtle/sprook 0 breder
.DEPOSIT 53257 1 (of 3) maakt Turtle/sprook 1 breder
.DEPOSIT 53258 1 (of 3) maakt Turtle/sprook 2 breder
.DEPOSIT 53259 1 (of 3) maakt Turtle/sprook 3 breder

Indien u met een andere dan de Atari 8-bits computer werkt, zoek dan de juiste POKE-waarden voor uw computer in uw handleiding op. De genoemde POKE-waarden zijn uiteraard dezelfde als die voor de Sprites worden gebruikt.

Om weer naar de juiste grootte terug te keren zet u in de genoemde geheugenplaatsen een 0.

Als we een sprook ontworpen hebben verlaten we de Shape Editor. Bij Atari doen we dit met de ESC-toets. Op dat moment zit de vorm opgeborgen in het werkgeheugen van de computer. Met de opdracht SETSH 1 (SETSHape) vertellen we de Turtle(s) die we het laatst aangeroepen hebben, de zojuist ontworpen vorm aan te nemen.

Het is verstandig om de sprook een naam te geven. Door het geven van een naam kennen we alle gegevens van de ontworpen sprook toe aan een variabele.

De opdracht die de gegevens voor ons uit de Shape Editor haalt is GETSH. Indien u een sprook ontworpen hebt en u typt:
PR GETSH 1, krijgt u een lijst met zestien getallen; dit zijn de gegevens van de eerste sprook.

Met de volgende opdracht kennen we de gegevens toe aan een variabele: MAKE "VORM GETSH 1 . In de variabele :VORM bergen we, eenvoudig gezegd, alle informatie op die nodig is om sprook 1 te maken.

Als we dit gedaan hebben, kunnen we de gegevens rustig wegschrijven op band/diskette of printer. Wanneer op een later tijdstip de gegevens weer van band/diskette ingelezen worden, moeten we de waarde van de variabele als het ware 'terughalen' naar de Shape Editor. Dat doen we met de opdracht PUTSH aldus:
PUTSH 1 :VORM .

Met GETSH halen we als het ware de gegevens dus uit de Shape

Editor en met PUTSH stoppen we de gegevens weer in de Shape Editor. U begrijpt nu wel, dat in de Logo-sprokenwereld EDSH, SETSH, GETSH en PUTSH heel belangrijke opdrachten zijn.

Het is hierboven al gezegd: een sprook kunt u beschouwen als een Turtle in een 'ander jasje'. Met een sprook kunt u dan ook dezelfde dingen doen als met de Turtle. Sproken kunt u:

- een kleur naar keus geven met SETC (gevolgd door een kleurnummer
- een snelheid naar keuze geven met SETSP (gevolgd door een snelheid. De invoer moet hier een getal tussen -200 en 200 zijn. Bij een negatieve snelheid beweegt de Turtle/sprook achteruit.

Opmerking. In tegenstelling tot de Turtle zelf (sprook 0 dus) kunnen we de vorm van een sprook niet draaien. Wel kunnen we een sprook van richting laten veranderen.

- besturen met bijvoorbeeld de joystick. Hiervoor is slechts een kleine procedure nodig waarbij we gebruik maken van SETH (SETHeading). Met SETH draaien we de Turtle/sprook in de aangegeven richting. De waarde van de variabele :richting wordt in de procedure bepaald door de uitvoer van de joystick in poort 0.

```
EDIT "BEWEEG
MAKE "richting JOY 0
IF :richting = "0 [SETH 0]
IF :richting = "2 [SETH 90]
IF :richting = "4 [SETH 180]
IF :richting = "6 [SETH 270]
WHEN 3 [STOP SETSP 0]
SETSP 50
BEWEEG
END
```

Opmerking. Sommige Logo's kennen deze procedure als een hulpprogramma op schijf.

U ziet het, deze recursieve procedure zorgt ervoor dat de sprook of de Turtle zich verplaatst en reageert op de beweging van de joystick. De uitvoer van de joystick kan zijn:

omhoog	0		
rechts-omhoog	1	'rustpositie'	-1
rechts	2		
rechts-beneden	3		
beneden	4		
links-beneden	5		
links	6		
links-omhoog	7		

Bovendien ziet u in de procedure de opdracht WHEN 3 [STOP SETSP 0] staan. Met WHEN activeren we een 'WHEN-DEMON'. Later komen we hierop terug. De bedoeling is in ieder geval dat de Turtle stopt als de vuurknop op de joystick wordt ingedrukt.

Nu willen we de Turtle/sprook besturen via het toetsenbord. Dat is heel eenvoudig:

```
TO BEWEEG2
MAKE "richting RC
IF :richting = "V [SETH 0]
IF :richting = "A [SETH 180]
IF :richting = "R [SETH 90]
IF :richting = "L [SETH 270]
IF :richting = "S [SETSP 0 STOP]
SETSP 50
BEWEEG2
END
```

U ziet, nu wordt de waarde van de variabele :richting bepaald door de RC-opdracht.

6.3 Een sproken-bibliotheek

Het kan erg handig zijn om een soort 'sproken-bibliotheek' aan te leggen: een verzameling sproken ingedeeld naar vorm en functie. Iedere keer bij het programmeren kunt u hier dan gebruik van maken. Hier volgt stap voor stap hoe u dit kunt doen.

Het opzetten van een sproken-bibliotheek

1. Ontwerp op papier enkele sproken. Gebruik hiervoor een ontwerpblad zoals achter in dit hoofdstuk is bijgevoegd.
2. Zet de sproken in de Shape Editor met de opdracht EDSH. Let op: u kunt bij Atari niet meer dan vijftien sproken tegelijk verwerken.
3. Geef de zojuist ontworpen sproken een naam met GETSH. Zorg dat u later aan de naam kunt zien welke sprook het is. Bijvoorbeeld: MAKE "auto.vooruit GETSH 1
   MAKE "auto.rechts GETSH 2
   enzovoort.
4. Sla de sproken op band op met SAVE "C: of op diskette met SAVE "D:bestandsnaam. Kies voor het bestand een herkenbare en met de bedoelde sprook verband houdende naam, bijvoorbeeld: SAVE "D:VOERTUIG.001.

5. Haal alle variabelen uit het geheugen op met ERNS (ERase NameS).
6. Type RECYCLE; dit verwijdert alle typefouten e.d. uit het geheugen.
7. Herhaal de stappen 1 t/m 6 zo vaak u wilt.

Op deze manier kunt u vele honderden sproken maken en op een diskette opslaan.

Als u later gebruik wilt maken van deze sproken, doet u het volgende:

- Kijk eerst welke sprokenverzamelingen beschikbaar zijn; met CATALOG "D: krijgt u een opgave van de diskette-inhoud.
- Kies een groep sproken en laad deze met LOAD "D:bestandsnaam of met LOAD "C: .
- Kijk met PONS welke sproken er in de geladen groep staan (PONS staat voor Print Out NameS).
- Kies de sproken die u wilt gebruiken.
- Verwijder de sproken die u niet gebruikt uit het geheugen met ERN :naam.
- Zet de gegevens van de te gebruiken sproken in de Shape Editor. Doe dit met de opdracht PUTSH, bijvoorbeeld: PUTSH 1 :auto.rechts.
- Als dit allemaal gedaan is, zijn de sproken gereed om gebruikt te worden.

Opmerking. Natuurlijk kunnen er veel meer dan vijftien sproken in een bestand staan. Het is echter nogal onoverzichtelijk om veel sproken in één bestand te zetten. Bovendien moeten er, als u er maar één of twee nodig hebt, dan ook zo veel verwijderd worden. In de praktijk blijkt een bestand met vijftien sproken juist handig in het gebruik te zijn.

6.4 Animatie met sproken

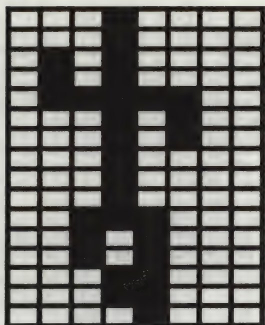
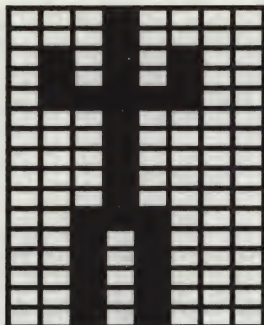
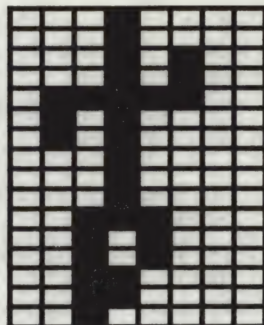
In §6.2 zagen we hoe sproken in beweging gezet kunnen worden met de opdracht SETSP, met de joystick of via het toetsenbord. Bijzonder interessant wordt het wanneer we animatie gaan toepassen. Animatie is de techniek van het snel achter elkaar laten zien van verschillende vormen (vergelijk tekenfilms), zodat 'beweging' wordt gesimuleerd.

Ook bij het toepassen van animatie bij sproken hoeven we niet veel anders te doen dan:

1. enkele sproken te ontwerpen die samen een 'beweging maken', sproken dus die veel op elkaar lijken;
2. deze sproken heel kort na elkaar te laten zien.



Dat lijkt heel eenvoudig nietwaar? En in feite is het dat ook. Kijkt u maar eens naar de volgende drie eenvoudige figuurtjes. Het zijn sproken zoals iedereen ze kan ontwerpen. Natuurlijk kunt u voor een animatie ook meer dan drie sproken gebruiken. Hoe meer sproken, hoe vloeiender de 'beweging'.

Er zijn verschillende manieren om de animatie in een procedure vast te leggen. Belangrijk is in ieder geval, dat de procedure recursief is. De beweging moet zich immers herhalen. Eventueel zou u van de opdracht REPEAT gebruik kunnen maken.

We nemen aan dat bovenstaande sproken 1, 2 en 3 genummerd zijn. Dan kunnen we de volgende opdracht samenstellen:

```
SETSH 1 WAIT 3 SETSH 2 WAIT 3 SETSH 3 WAIT 3
```

Wanneer deze belangrijke opdracht in een recursieve procedure komt te staan, gebeurt het volgende. De Turtle neemt vorm 1 aan, wacht 1/20 seconde (= 3/60), neemt vorm 2 aan, wacht weer, neemt vorm 3 aan, wacht weer, en neemt dan, omdat de procedure recursief is, opnieuw vorm 1 aan.

Dit zou zo'n procedure kunnen zijn:

```
EDIT "ANIMATIE.1
TELL 0 ST PU SETSP 50
SETSH 1 WAIT 3
SETSH 2 WAIT 3
SETSH 3 WAIT 3
ANIMATIE.1
END
```

Eleganter is natuurlijk:

```
MAKE "vorm  1

EDIT "ANIMATIE.1
IF :vorm = "4 [MAKE "vorm 1]
TELL 0 ST PU SETSP 50
SETSH :vorm WAIT 3
MAKE "vorm :vorm + 1
ANIMATIE.1
END
```

U begrijpt dat beide procedures precies dezelfde werking hebben. De laatste procedure is vooral geschikt voor animatie met 'veel' sproken.

Wilt u een animatie met, laten we zeggen, vijf sproken maken, dan kan het ook zo:

```
EDIT "ANIMATIE.1
MAKE "vorm 1
REPEAT 5 [SETSH :vorm MAKE "vorm :vorm + 1 WAIT 3]
MAKE "vorm 4
REPEAT 4 [SETSH :vorm MAKE "vorm :vorm - 1 WAIT 3]
ANIMATIE.1
END
```

Ziet u wat in deze laatste procedure bijzonder is? De volgorde waarin de vormen op het beeld worden gebracht is:

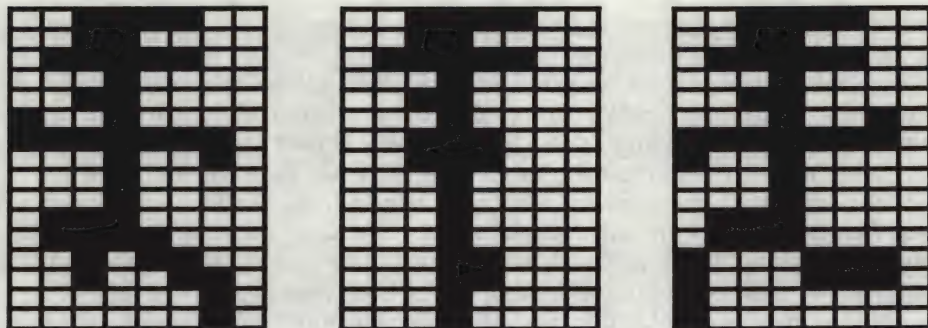
1 2 3 4 5 4 3 2 1 enzovoort

en niet, zoals bij eerdere procedures:

1 2 3 1 2 3 enzovoort.

U ziet dat de mogelijkheden erg uitgebreid zijn en dat de sproken wel bijzonder gebruikersvriendelijk zijn.

Kijkt u eens naar deze drie sproken:



Gecombineerd leveren zij een aardig plaatje van een 'jogger' op. Denkt u eraan: u moet eerst LT 90 intypen voordat u de jogger in beweging zet. De figuurtjes suggereren namelijk een beweging van rechts naar links. U kunt LT 90 niet in de procedure zetten, want dan zal de jogger, omdat de procedure recursief is, blijven ronddraaien.

Erg handig bij het maken van sproken voor een animatie is de procedure COPYSH. COPYSH, te gebruiken als een opdracht (een Logo-primitief), kopieert een vorm. En dat is erg gemakkelijk. Zoals u ziet zijn de verschillen bij bovenstaande sproken erg klein. Het verschil zit in de armen en de benen. Wat doen wij dus:

- sprook 1 ontwerpen
- COPYSH 1 2 intypen
- sprook 2 zijn eigen vorm geven
- COPYSH 1 3 intypen
- sprook 3 zijn eigen vorm geven.

COPYSH levert een aanzienlijke tijdwinst op, zeker bij het maken van meer sproken, en blijkt een handige hulpprocedure te zijn. U vindt de listing verderop in dit hoofdstuk.

Sproken worden nog interessanter wanneer u 'WHEN-DEMONS' inschakelt. Denk eraan: dit kan lang niet bij elke Logo-versie.

6.5 Over 'Collision' en 'When-Demons'

Hoewel When-Demons niet exclusief voor Atari Logo zijn, kunnen we toch vaststellen dat slechts weinig Logo-versies over deze mogelijkheid beschikken. In de Nederlandstalige MSX Logo worden ze 'demonen' genoemd. Wat zijn When-Demons dan precies?

U zou de When-Demons kunnen bekijken als een soort 'oppassers' in de wereld van de sproken. Eenmaal geactiveerd blijft een When-Demon waakzaam. Een When-Demon wacht ..... Hij wacht op iets heel specifieks.

De 21 beschikbare When-Demons wachten allemaal op iets anders. Ieder heeft zijn eigen taak. Er is bijvoorbeeld een When-Demon die wacht totdat de vuurknop (van de joystick) wordt ingedrukt. Er is ook een When-Demon die oplet of Turtle 0 wellicht in aanraking komt met een lijn, getrokken door pen nummer 0. Er is een andere When-Demon die pas in actie komt als Turtle 0 en Turtle 1 met elkaar in botsing komen. Er is zelfs een When-Demon die, eenmaal geactiveerd, iedere seconde zijn opdracht uitvoert.

Zolang er niets gebeurt zijn de When-Demons ogenschijnlijk inactief, maar vergis u niet. Zodra zich een gebeurtenis voordoet waar een When-Demon voor geactiveerd is, treedt hij in werking. De When-Demon doet dan wat u hem heeft opgedragen.

Als voorbeeld nemen we de, hierboven al genoemde, When-Demon die op de vuurknop van de joystick let. We activeren hem door in te typen:

```
WHEN 3
```

Deze When-Demon is nummer 3. Deze activeringsopdracht is echter niet volledig. Als we een When-Demon activeren moeten we ook zorgen voor een opdracht; een opdracht die moet worden uitgevoerd als de genoemde When-Demon zijn gebeurtenis bespeurt. Een volledige activeringsopdracht kan dus zijn:

```
WHEN 3 [PR [Nu heeft u dus de vuurknop ingedrukt]]
```

Het kan ook anders:

```
WHEN 3 [ST PD REPEAT 4 [FD 100 RT 90]]
```

Het maakt allemaal niet uit.

Opmerking. De opdracht WHEN kan slechts worden gebruikt bij SS (Split Screen) of FS (Full Screen).

U kunt de When-Demons allerlei soorten opdrachten geven: tekenopdrachten, printopdrachten, geluidsopdrachten, animatieopdrachten, enzovoort. Deze opdrachten kunt u aan 21 verschillende When-Demons geven.

De When-Demons zijn te verdelen in drie groepen:

1. When-Demons die letten op een confrontatie (ook wel botsing of 'Collision' genoemd) tussen een Turtle en een getekende lijn;
2. When-Demons die letten op een confrontatie tussen twee Turtles, of sproken natuurlijk;
3. When-Demons die letten op wat genoemd wordt: NON TURTLE EVENTS. Voorbeelden zijn When-Demons die letten op: de vuurknop, positie van de joystick. We noemden ook reeds de When-Demon die, eenmaal geactiveerd, iedere seconde zijn opdracht uitvoert.

Zoals u ziet is deze laatste When-Demon wel erg actief. Het is een aparte When-Demon met vele mogelijkheden. Wat zou u zeggen van een secondenteller:

```
SS MAKE "X 0 WHEN 7 [PR :X MAKE "X :X + 1 CT]
```

Deze 'one-liner' maakt op uw beeldscherm een secondenteller. Toegepast in een procedure maakt u uw eigen Logo-klok.

Lijst van When-Demons

Opmerking. Zoals u ziet is de lijst gemaakt in vier kolommen. In de eerste kolom staat het nummer van de When-Demon. In de tweede en de derde kolom staat op welke confrontatie de When-Demon let. Dit kan zijn: een confrontatie tussen een Turtle en een pen of tussen twee Turtles onderling. In de vierde kolom staan de 'Non Turtle Events', of wat er moet gebeuren om de When-Demons nr. 3, 7 en 15 te activeren. Om redenen die de auteur van dit boek niet bekend zijn wordt When-Demon nr.11 niet gebruikt.

De lijst is afgedrukt op de volgende pagina.

Opmerking. Het is mogelijk om meer When-Demons tegelijk te activeren. Ze zullen echter niet tegelijk actief zijn.
Hun oplettendheid en activiteit wordt bepaald door hun kracht.
When-Demon 0 is de sterkste en daarmee ook de snelste 'Demon'.
When-Demon 21 is de minst sterke en daarmee ook de traagste.

Lijst van When-Demons

When-Demon:	Turtle:	Pen:	Gebeurtenis:
0	0	0	
1	0	1	
2	0	2	
3			Vuurknop in
4	1	0	
5	1	1	
6	1	2	
7			Elke seconde
8	2	0	
9	2	1	
10	2	2	
11			Niet gebruikt
12	3	0	
13	3	1	
14	3	2	
15			Joystick pos.

When Demon:	Turtle:	Turtle:
16	3	0
17	3	1
18	3	2
20	0	1
21	0	2
22	1	2

Nu willen we weten wat we met deze When-Demons kunnen doen en hoe we ze in een procedure kunnen activeren. Kijkt u eens naar dit eenvoudige voorbeeld:

```
TO LIJN
TELL 0 ST PU FS
FD 50 RT 90 PD FD 400 PU
HOME
BK 50 RT 90 PD FD 400 PU
HOME
MAKE "X 0
STARTTURTLE
END

TO STARTTURTLE
IF :X = "10 [SETSP 0 HOME STOP]
TELL 0 ST PU
SETSP 100
WHEN 0 [RT 180 MAKE "X :X + 1]
STARTTURTLE
END
```

U weet wat er gebeurt: Turtle 0 tekent boven en onder op het scherm een lijn, maar zal deze nooit passeren. De geactiveerde When-Demon (nr.0) zorgt er namelijk voor, dat de Turtle zich, na het aanraken van de lijn, 180 graden draait. Eveneens verhoogt de When-Demon de waarde van :X met 1. U begrijpt dat :X een functie heeft als teller. Zodra de waarde van :X = 10 is, stopt de recursieve procedure.

Om de When-Demon te activeren hoeven we niet gebruik te maken van een recursieve procedure. Eenmaal geactiveerd blijft de When-Demon zijn werk doen. De werking noemen we dan ook 'global' (algemeen) en niet 'local' (plaatselijk). Als we de When-Demon in de niet-recursieve procedure LIJN hadden geactiveerd, was de werking gelijk geweest. We maken echter voor de teller gebruik van recursie.

Ook in het volgende voorbeeld wordt gebruik gemaakt van recursie, uitsluitend voor de testopdrachten:

```
TO ALCATRAZ
PD REPEAT 4 [FD 75 RT 90]
PU FD 50 RT 90 FD 50 LT 90
MAKE "X 31 MAKE "Y 0
LOOPTURTLE
END
```

```
TO LOOPTURTLE
IF :X = "24 [MAKE "X 31]
IF :Y = "50 [SETSP 0 STOP]
WHEN 0 [RT ( RANDOM 30 ) + 150]
WHEN 7 [SETC :X MAKE "X :X - 1 MAKE "Y :Y + 1]
SETSP 50
LOOPTURTLE
END
```

In de procedure ALCATRAZ wordt een vierkant getekend waaruit de Turtle niet kan ontsnappen. Tevens wordt hier aan de variabele :X (de kleur van de Turtle) de waarde 31 en aan de variabele :Y (de teller) de waarde 0 toegekend. Dan wordt de recursieve procedure aangeroepen.

LOOPTURTLE controleert eerst de waarde van beide variabelen.
Is de waarde van :X al 24? Zet dan de waarde weer op 31!
Is de waarde van :Y al 50? Zet dan de procedure maar stop!

Daarna worden er twee When-Demons geactiveerd:

nr.0 draait de Turtle tussen de 150 en 180 graden rechtsom wanneer de Turtle in aanraking komt met de lijnen van het vierkant.

nr.7 - verlaagt iedere seconde de waarde van :X met 1. Het gevolg hiervan is, dat de Turtle, van boosheid, steeds roder wordt.
- verhoogt iedere seconde de waarde van :Y met 1. Het gevolg hiervan is, dat de ontsnappingspogingen van de Turtle plaatsvinden binnen de vastgestelde tijdsduur (in dit geval 50 seconden).

Tenslotte wordt de Turtle in beweging gezet door SETSP 50.

Ziet u welke aardige dingen mogelijk zijn met de When-Demons? Natuurlijk is het onmogelijk op alle facetten in te gaan. U zult zelf moeten experimenteren.

Nog een voorbeeld?

```
EDIT "VELD
PU FD 90 RT 90 PD
FD 90 RT 90 REPEAT 3 [FD 180 RT 90]
FD 90 PU
FS HOME
TELL [0 1 2 3]
EACH [ST SETH 90 + WHO SETSP 50 PU]
MAKE "tijd 0
TURTLEDANS
END
```

```
EDIT "TURTLEDANS
IF :tijd = "60 [SETSP 0 STOP HOME]
WHEN 0 [ASK 0 RT 180]
WHEN 4 [ASK 1 RT 180]
WHEN 8 [ASK 2 RT 180]
WHEN 12 [ASK 3 RT 180]
WHEN 16 [ASK 3 RT RANDOM 180]
WHEN 17 [ASK 1 RT RANDOM 180]
WHEN 18 [ASK 2 RT RANDOM 180]
WHEN 19 [ASK 0 RT RANDOM 180]
WHEN 20 [ASK 2 RT RANDOM 180]
WHEN 21 [ASK 1 RT RANDOM 180]
WHEN 7 [MAKE "tijd :tijd + 1]
TURTLEDANS
END
```

Tot zover de voorbeelden van de When-Demons. Zoals we eerder zagen kunt u met deze technieken uw eigen spelen maken. We zien dit in: Super Shuttle.

6.6 *Een voorbeeldspel: Super Shuttle*

Dit spel, dat uit één hoofd- en elf subprocedures bestaat, is een voorbeeld van wat er zoal met sproken en When-Demons gedaan kan worden. Het is een basisspel; iedereen kan er naar eigen idee veranderingen in aanbrengen. Het spel kan ook op eenvoudige wijze gemakkelijker of moeilijker worden gemaakt. Bij de bespreking van de procedures wordt dit wel aangegeven.

Het idee achter het spel is:

-- Als piloot van een geavanceerd ruimteveer, de Super Shuttle,
-- vliegt u door het oneindige heelal. Het is uw taak om tijdens de
-- vlucht met de lasers van uw schip de rondvliegende meteorieten
-- te vernietigen, anders zouden deze meteorieten grote schade aan
-- het vliegverkeer en de ruimtestations kunnen toebrengen.

Als u het spel start (met SPEL) wordt eerst het beeldscherm opgebouwd. Er ontstaat een heldere sterrenhemel tegen een zwarte achtergrond. Dan verschijnt de maan, en even later de Shuttle en de meteorieten. Al deze figuren zijn sproken.

Met de joystick beweegt u uw schip door de ruimte. Indien er zich voor u een meteoriet bevindt, schakelt u de laser is met een druk op de vuurknop. Op dat moment wordt razendsnel uw Y-positie en die van de meteoriet bepaald. Is deze, met een marge van 10, gelijk, dan zal de meteoriet exploderen.

Door de marges van de Y-posities nu kleiner/groter te maken,

maakt u het spel moeilijker/gemakkelijker.

Ook de explosies (twee verschillende) worden voorgesteld door sproken. Door de twee verschillende explosies ieder hun eigen kleur te geven vormen ze, achter elkaar geprojecteerd, een prachtige tweekleurige sprook.

Als uw vliegtijd (die u zelf kunt instellen) voorbij is, kunt u lezen hoe vaak u een meteoriet heeft vernietigd.

Let op. Voor andere dan Atari 8-bits Logo-versies zullen enkele dingen moeten worden aangepast:

- .DEPOSIT 53275 1 om Turtle 1 groter te maken. Voor een andere Logo zal nu een andere waarde op een andere geheugenplaats moeten worden gezet. Deze opdracht kan echter ook weggelaten worden!
- De kleuropdrachten SETC en SETBG zullen andere waarden moeten hebben.
- Ook de specifieke sprookopdrachten zullen moeten worden aangepast.

Kijk hier goed naar voordat u het spel start.

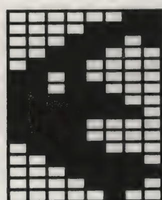
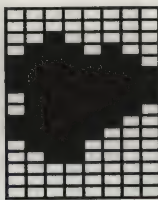
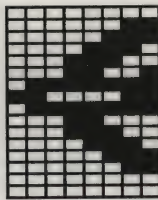
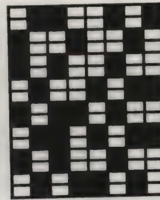
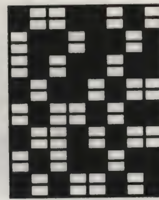
Super Shuttle

Gebruikte variabelen:

:BOTS	= het aantal keren dat u een meteoriet hebt vernietigd.
:TIJD	= de speeltijd / ingesteld op 200 (x 1/60 seconde).
:P1	= de (INT) Y-coördinaat van Turtle 1.
:P2	= de (INT) Y-coördinaat van Turtle 2.
:P3	= de (INT) Y-coördinaat van Turtle 3.
:ANTWOORD	= het teken voor wel/niet (J/N) nog eens spelen van het spel.

Gebruikte sproken:

Hier ziet u de afbeeldingen van de gebruikte sproken. Achtereenvolgens: maan, steen, shuttle, explosie.1 en explosie.2.

Toets dus naast de procedure tevens in:

```
MAKE "explosie.2 [154 154 109 109 91 91
180 180 75 75 148 148 90 90 165 165]
MAKE "explosie.1 [101 101 146 146 164 16
4 75 75 180 180 107 107 164 164 90 90]
MAKE "SHUTTLE [0 3 7 14 28 25 127 195 12
7 24 28 14 7 3 0 0]
MAKE "STEEN [0 0 32 116 254 255 255 127
126 252 248 112 48 0 0 0]
MAKE "MAAN [3 15 29 57 112 208 216 252 2
54 240 240 120 62 31 14 4]
```

Gebruikte kleuren:

- achtergrond : zwart
- shuttle : oranje/geel
- maan : goudkleur
- steen : lichtblauw
- explosie 1 : wit
- explosie 2 : rood/oranje

Gebruikte procedures:

1. SPEL : de hoofdprocedure; met het aanroepen van deze procedure begint het spel. In SPEL worden enkele variabelen op 0 gezet en worden de subprocedures HEMEL, METEORIETEN en SPACE. SHUTTLE aangeroepen.
2. HEMEL : In deze procedure wordt met behulp van STERREN een sterrenhemel getekend. Ook verandert Turtle 0 hier in een maan en deze wordt in positie gebracht.
3. STERREN : tekent met behulp van de procedures DOT1 en DOT een sterrenhemel met 100 sterren.
4/5. DOT1 en DOT: plaatsen op willekeurige coördinaten een stip.
6. METEORIETEN : een belangrijke procedure die achtereenvolgens de Turtles 2 + 3 tot een STEEN maakt, ze een kleur, een willekeurige positie, richting en snelheid geeft.
7. SPACE.SHUTTLE : Hier krijgt Turtle 1 de vorm van een shuttle. De shuttle krijgt kleur, richting en snelheid. Tenslotte wordt de recursieve procedure VLIEGEN aangeroepen.
8. VLIEGEN : Dit is de recursieve procedure die de shuttle met de joystick bestuurbaar maakt. Door deze besturing aan te passen kunnen we het spel moeilijker maken. Ook wordt hier When-Demon

		nr.3 geactiveerd. When-Demon 3 let erop of de vuurknop van de joystick wellicht wordt ingedrukt. Als dit gebeurt treedt de procedure MM in werking.
9. MM	:	Door deze procedure wordt razendsnel de positie van de shuttle (Turtle 1) en van de meteorieten (Turtles 2 + 3) uitgerekend. Als op het moment dat de vuurknop wordt ingedrukt de Y-positie van de shuttle niet groter dan de Y-positie + 10 of kleiner dan de Y-positie - 10 van een van beide meteorieten is, treedt B2 of B3 in werking. Maak om het spel moeilijker te maken de marges kleiner (bijvoorbeeld van 10 naar 5).
10/11. B2/B3	:	de 'explosie'-procedures. De Turtles 2 en/of 3 veranderen hier van een meteoriet in een explosie. Ook wordt hier de waarde van :BOTS met 1 opgehoogd. Na dit alles verandert de Turtle van explosie gewoon terug in een meteoriet.
12. SLOT	:	Het einde van het spel. Hier wordt vermeld hoeveel meteorieten geraakt zijn, en krijgt de speler de kans het spel nogmaals te spelen.

Dat waren ze, alle twaalf de procedures van Super Shuttle. Zoals gezegd is het spel betrekkelijk eenvoudig. U kunt, met wat fantasie, echter dit spel gebruiken als uitgangspunt voor uw eigen versie van Super Shuttle. U kunt het spel ook zien als een introductie in het zelf-spelen-maken met de mogelijkheden die Logo u biedt. Wat u er ook mee doet, hier komt de listing van de twaalf procedures.

SUPER SHUTTLE

```
TO SPEL
CS FS
TELL [0 1 2 3] HT PU
MAKE "BOTS 0
MAKE "TIJD 0
HEMEL
METEORIETEN
SPACE.SHUTTLE
END


TO HEMEL
TELL 0 PD SETBG 0
STERREN 100
PUTSH 1 :MAAN
TELL 0 SETSH 1 ST SETC 14
PU SETPOS [-120 90]
END
```

```
TO STERREN :TELLER
IF :TELLER = "0 [STOP]
DOT LIST RANDOM 300 RANDOM 200
STERREN :TELLER - 1
END


TO DOT1 :OLDPOS :POS :PEN :SHOWNP
HT PU
SETPOS :POS SETPC 0 14
PD FD 0 PU SETPOS [0 0]
RUN FPUT :PEN []
IF :SHOWNP [ST]
END


TO DOT :POS
ASK FIRST WHO [DOT1 POS :POS PEN SHOWNP]
END


TO METEORIETEN
PUTSH 3 :STEEN
TELL [2 3] SETSH 3 SETC 78 PU
SETX ( RANDOM 200 ) - 100
SETY ( RANDOM 200 ) - 100
EACH [SETH ( RANDOM 180 ) + 90]
EACH [SETSP ( RANDOM 50 ) + 25]
ST
END


TO SPACE.SHUTTLE
PUTSH 2 :SHUTTLE
.DEPOSIT 53257 1
TELL 1 SETSH 2 SETC 125 ST
SETH 270 SETSP 199
VLIEGEN 0
END
```

```
TO VLIEGEN :TIJD
IF :TIJD > 200 [SLOT STOP]
TELL 1 SETH 270
PUTSH 4 :explosie.1
PUTSH 5 :explosie.2
MAKE "RICHTING JOY 0
IF :RICHTING = "0 [RT 90 FD 25 LT 90]
IF :RICHTING = "4 [LT 90 FD 25 RT 90]
IF :RICHTING = "6 [FD 50]
IF :RICHTING = "2 [BK 50]
WHEN 3 [MM]
ERN "RICHTING
VLIEGEN :TIJD + 1
END

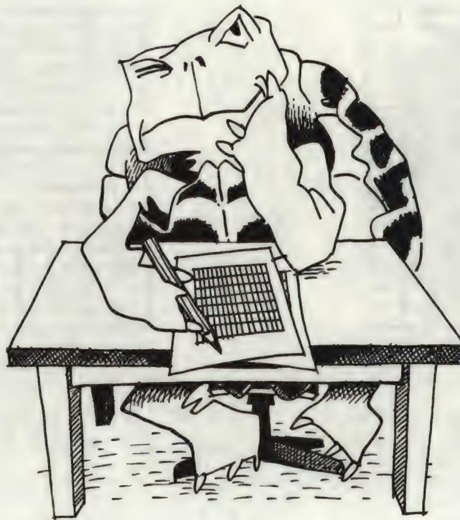

TO MM
TELL 2 MAKE "P2 INT YCOR
TELL 3 MAKE "P3 INT YCOR
TELL 1 MAKE "P1 INT YCOR
IF AND :P1 < :P2 + 10 :P1 > :P2 - 10 [B2]
IF AND :P1 < :P3 + 10 :P1 > :P3 - 10 [B3]
END


TO B2
TELL 2 SETSP 0 SETSH 4 SETC 7
WAIT 50 HT
MAKE "BOTS :BOTS + 1
METEORIETEN
END

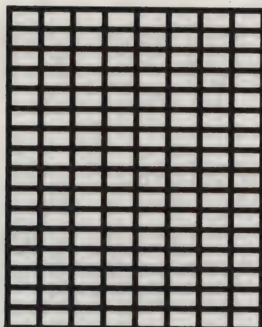

TO B3
TELL 3 SETSP 0 SETSH 5 SETC 24
WAIT 50 HT
MAKE "BOTS :BOTS + 1
METEORIETEN
END
```

```
TO SLOT
TELL [O 1 2 3] SETSP O SETSH O HOME HT
TS CT
PR [Gefeliciteerd .......]
PR []
PR []
PR [In deze vliegreis met]
PR [de Super Shuttle ,]
PR ( SE [heb je in totaal :] :BOTS
PR [keer een meteoriet vernietigd !]
PR [Nog eens proberen ?]
PR []
PR [Toets maar J / N]
MAKE "ANTWOORD RC
IF :ANTWOORD = "J [CT SPEL] [CT]
END
```

6.7 *Sproken voor gevorderden*



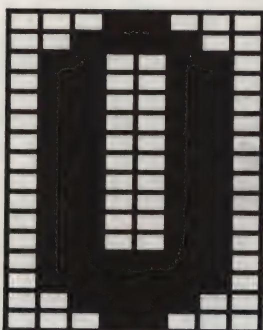
Zoals u reeds hebt gezien worden de sproken ontworpen in de Shape Editor. Wanneer u de Shape Editor aanroept krijgt u het raster te zien zoals dat op de volgende bladzijde is afgebeeld. Er zijn zestien horizontale rijen; iedere rij is één byte informatie. Zoals u weet bestaat een byte uit acht bits en kijk: op iedere rij staan acht hok-



jes (nr.0-7) te wachten om te worden ingevuld.

Vult u bij het maken van een sprook zo'n hokje in, dan krijgt dit bit de waarde 1. Blijft een hokje open, dan krijgt het desbetreffende bit de waarde 0.

Laten we maar eens gaan kijken naar een voorbeeld. Het eerste plaatje is een vorm die als voorbeeld dient. Het tweede plaatje is het bit-patroon van de vorm. Bij de ingekleurde hokjes heeft een bit de waarde 1, bij de lege/open hokjes de waarde 0.



0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0

Bekijken wij het bitpatroon van de vorm zorgvuldig, dan zien we op iedere horizontale rij een binair getal staan. In het voorbeeld zijn dat de volgende getallen:

Rij 0 : 00011000
Rij 1 : 00111100
Rij 2 : 01100110
Rij 3 : 01100110
Rij 4 : 01100110
Rij 5 : 01100110
Rij 6 : 01100110
Rij 7 : 01100110
Rij 8 : 01100110
Rij 9 : 01100110
Rij 10 : 01100110
Rij 11 : 01100110
Rij 12 : 01111110
Rij 13 : 00111100
Rij 14 : 00111100
Rij 15 : 00011000

Als u hoofdstuk 4: Logo cijfers en getallen heeft doorgelezen, zal het u geen moeite kosten deze binaire getallen om te zetten in decimale. Als u dat doet krijgt u de volgende decimale getallen:

24 60 102 102 102 102 102 102 102 102 102 126 60 24

En dit zijn de gegevens voor de vorm van de vorige bladzijde. Wanneer u deze vorm met EDSH 1 zou maken en u typt PR GETSH 1, dan krijgt u de zojuist genoemde getallen.

Overigens hoeft u de binaire waarden niet om te rekenen om de decimale waarden te weten. Als we boven het raster uit de Shape Editor de binaire machten schrijven, kunt u gewoon optellen. Kijk maar:
Rij 0 = 1 x 16 + 1 x 8 = decimale waarde: 24
Rij 1 = 1 x 32 + 1 x 16 + 1 x 8 + 1 x 4 = decimale waarde: 60
Op deze manier kunt u, zonder binair/decimaal omrekenen, zelf de decimale waarden van de vorm uitrekenen.

128 64 32 16 8 4 2 1 = waarde
7 6 5 4 3 2 1 0 = bitnummer
rij 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Op dezelfde manier als wij in de Logo-sprokenwereld vormen en figuren maken, is ook de tekenverzameling van uw computer ontwikkeld. Ergens in het geheugen van uw computer liggen de binaire gegevens opgeslagen van alle tekens waarover uw computer beschikt. De letters, de cijfers, maar ook : ! " # $ % & ' @ ( ) < > [ ] ? enzovoort.

Het zou plezierig zijn als we deze gegevens konden gebruiken om sproken mee te maken. Sproken in de vorm van letters, sproken in de vorm van cijfers. Daar zijn veel aardige sprokenprocedures mee te schrijven. Wat denkt u bijvoorbeeld van vier sproken die samen woorden maken? Of van vier sproken die samen de digitale tijd aangeven?

Laten we eens kijken of we dat voor elkaar kunnen krijgen. Op de Atari 8-bits computers staat de tekenverzameling in de lokaties 57344 - 58367. Werkt u met een andere computer dan moet het niet al te moeilijk zijn om de desbetreffende geheugenplaatsen in uw

machine te vinden. Kijk daarvoor in uw handboek.

Nu is de tekenverzameling van de Atari niet opgebouwd uit zestien bytes (zoals de sproken bij Logo), maar uit acht bytes. Wanneer we dus achter de gegevens van een teken willen komen, zullen we acht bytes, ofwel acht geheugenplaatsen, moeten bekijken. Maar waar vinden we nu de gegevens van bijvoorbeeld het teken A?

Om de juiste geheugenplaats te vinden is nogal wat rekenwerk nodig. De eerste informatie die we nodig hebben is de ASCII-code van het gewenste teken.

Opmerking. De ASCII-code is de American Standard Code for Information Interchange. Ieder teken op het toetsenbord van uw computer heeft een code, die voor (bijna) alle computers gelijk is.

De decimale ASCII-code van A is 65. En dan de berekening. Kijkt u naar het volgende rijtje:

ASCII-code	Berekening
0 t/m 31	(ASCII-code + 64) x 8
32 t/m 95	(ASCII-code - 32) x 8
95 t/m 127	ASCII-code x 8

Voor het teken A wordt de berekening dan:

65 - 32 = 33
33 x 8 = 264

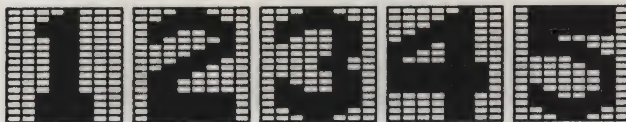
Het aldus verkregen getal is erg belangrijk.

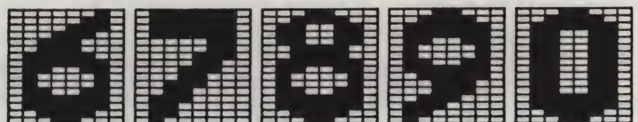
Tellen we dit getal op bij 57344 (dit is de geheugenlokatie waar de tekenverzameling begint), dan krijgen we 57608. En dit is het adres waar de informatie over de A begint.

Om alle informatie over de A te weten te komen zullen we echter niet één, maar acht adressen moeten bekijken. Immers de informatie van ieder teken ligt opgeslagen in acht bytes. Zo zijn dus op de geheugenadressen 57608 t/m 57615 de gegevens te vinden van de A.

Op deze manier kunt u nu zelf de gegevens van ieder teken vinden. Kijk daarvoor steeds eerst naar de ASCII-code van het teken en pas dan de juiste berekening toe.

Nogmaals: de genoemde geheugenadressen zijn die van de Atari 8-bits computers. Indien u met een andere computer werkt, zult u eerst de juiste geheugenlokaties moeten opzoeken!





Om nu met de gegevens van een teken een sprook te maken moeten we het volgende doen:

1. Het ASCII-codenummer van het teken bepalen.
2. Met behulp van dit codenummer de berekening uitvoeren.
3. Het aldus verkregen getal optellen bij 57344 (voor Atari 8-bits computers).
4. Uit de aldus verkregen geheugenplaats met .EXAMINE de waarde lezen.
5. Ook de waarde lezen van de zeven geheugenplaatsen daarna.
6. Deze waarden steeds tweemaal (een sprook heeft geen acht maar zestien bytes informatie nodig) in een lijst plaatsen.
7. Alle gegevens (in een lijst) toekennen aan een variabele.
8. De gegevens gebruiken om van een sprook een teken te maken.

Een en ander hoeft u niet zelf te doen. De volgende procedures nemen u alle rekenwerk uit handen en laten u in een fractie van een seconde de Turtle zien als door u gekozen teken. Als u bijvoorbeeld typt: MAKESH "B ziet u razendsnel Turtle 0 in een B veranderen.

Met behulp van deze procedures kunt u naar hartelust experimenteren.

```
TO MAKESH :N
SS SETBG 0 TELL 0 ST
ERN "VORM
MAKE "A ASCII :N
IF OR :A < 0 :A > 127 [ERN "A MAKESH :S
:N]
IF :A < 32 [MAKE "P ( :A + 64 ) * 8]
IF AND :A > 31 :A < 96 [MAKE "P ( :A - 3
2 ) * 8]
IF :A > 95 [MAKE "P ( :A * 8 )]
MAKE "PP :P + 57344
MAKE "BYTE 0
MAKE "VORM []
GET :PP
PUT
END

TO GET :PP
MAKE "GEGEVENS1 .EXAMINE :PP
MAKE "GEGEVENS2 .EXAMINE :PP
MAKE "VORM LPUT :GEGEVENS1 :VORM
MAKE "VORM LPUT :GEGEVENS2 :VORM
IF :BYTE = "7 [STOP]
MAKE "BYTE :BYTE + 1
GET :PP + 1
END

TO PUT
PUTSH 1 :VORM
TELL 0 ST SETSH 1
END
```

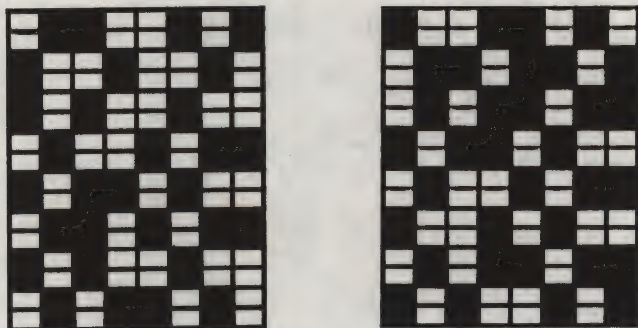
6.8 Het maken van meerkleurige sproken

Zoals we reeds eerder zagen kunnen we een sprook een kleur geven met SETC gevolgd door een kleurnummer. Wellicht heeft u zich, terwijl u dit hoofdstuk las, ook afgevraagd of het mogelijk is sproken in meerdere kleuren te maken. Het zal u teleurstellen, maar zowel in Atari LCSI Logo als in de meeste andere Logo-versies is het niet mogelijk om één sprook in meerdere kleuren af te beelden.

Het is evenmin mogelijk animatie te doen met één sprook. Om beweging te simuleren hebben we meerdere sproken nodig, en, u raadt het al, ook om meerkleurige sproken te maken kunnen we een soortgelijke techniek gebruiken. Bij animatie worden verschillende sproken na elkaar op het beeldscherm gebracht, en om meerkleurige sproken te maken worden ze tegelijkertijd (en achter elkaar) afgebeeld.

Wanneer we meerdere (in Atari LCSI Logo maximaal vier) sproken achter elkaar afbeelden en we geven iedere sprook een andere kleur, dan zien we een sprook met meerdere kleuren. Een mooi voorbeeld is de explosie die gebruikt wordt in het spel Super Shuttle. Door nu twee verschillende sproken te ontwerpen, ze ieder een andere, contrasterende kleur te geven en ze achter elkaar af te beelden, krijgen we een tweekleurige explosie.

Hier zijn de twee sproken die gebruikt zijn voor de explosie. Ziet u dat ze 'complementair' zijn, dat wil zeggen dat ze elkaar aanvullen? Het heeft geen zin ze elkaar te laten overlappen.

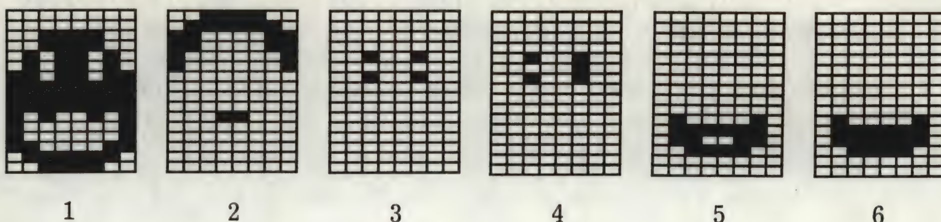


Ook bijvoorbeeld een gezicht is op deze wijze vrij nauwkeurig te maken. Natuurlijk kunnen we ook bij meerkleurige sproken met animatie werken. Dit gezicht is er een voorbeeld van. Er worden meerdere sproken in gebruikt: 1. het gezicht; 2. het haar en de snor; 3. de ogen open; 4. één oog dicht; 5. de mond open; 6. de mond dicht. Door nu achter elkaar (tegelijkertijd) sprook 1, 2, 3 en 5 af te beelden ontstaat een gezicht in vier kleuren.

Dit gezicht biedt ook animatiemogelijkheden:

- Laat om een knipoog te simuleren sprook 3 en 4 elkaar afwisselen.
- Laat om praten te simuleren sprook 5 en 6 elkaar afwisselen.

Hier zijn de zes benodigde sproken:



6.9 Nuttige hulpprogramma's

Zoals u in dit hoofdstuk heeft kunnen lezen heeft LCSI Logo van Atari vier opdrachten die specifiek gericht zijn op het werken met sproken. Het zijn:

1. EDSH : roept de Shape Editor aan. Deze opdracht wordt gebruikt bij het ontwerpen van sproken.
2. SETSH : geeft de laatst aangeroepen Turtle(s) de genoemde vorm.
3. GETSH : geeft als uitvoer een lijst met 16 getallen, zijnde de gegevens van de genoemde vorm.
4. PUTSH : geeft aan de genoemde vorm de, in een variabele vastgelegde gegevens.

De opdrachten worden in de volgende volgorde gebruikt: met EDSH 1 ontwerpt u de eerste vorm, met MAKE "v GETSH 1 geeft u aan de variabele "v (of een andere naam) de gegevens van de zojuist ontworpen vorm 1. In een procedure of in de 'Direct Mode', toetst u PUTSH 1 :v in; daarmee geeft u de gegevens uit de variabele :v aan vorm 1. De vorm 1 kan nu worden gebruikt: met SETSH 1 geeft u de laatst aangeroepen Turtle(s) deze gedaante.

Soms is het echter plezierig om een paar opdrachten meer ter beschikking te hebben wanneer u met sproken werkt; opdrachten om gemaakte sproken te kopiëren en te veranderen; opdrachten om gemaakte sproken te wissen of in een animatie te gebruiken. Over zulke opdrachten beschikken de meeste Logo's echter niet. Dat is geen probleem, we schrijven de opdrachten zelf!

Als voorbeeld vindt u elf voorbeeldprocedures die zich in Logo gedragen als opdrachten. Omdat de procedures als (zelfgedefinieerde) opdrachten zullen worden gebruikt, zijn voor de procedurenamen Engelse woorden gekozen. Werkt u echter met een Nederlandse

Logo dan is het handiger om ook Nederlandse procedurenamen te kiezen.

Opmerking. Denkt u erom, dat de procedures geschreven zijn voor de Atari LCSI Logo-versie? Dit is vooral belangrijk voor de procedures SYSRESET, ERSH, ALL en ALLSH. Iedere procedure is echter aan te passen aan elke Logo-versie.

Hier volgen eerst de procedures, later zullen we een en ander toelichten.

```
TO SYSRESET
SETBG 74 SETPC 0 15
SETPC 1 47 SETPC 2 121
TELL 0 SETC 7 TELL 1 SETC 20
TELL 2 SETC 44 TELL 3 SETC 68
.DEPOSIT 53256 0 .DEPOSIT 53257 0
.DEPOSIT 53258 0 .DEPOSIT 53259 0
TELL ALL SETSH 0 SETSP 0 CS PD HT
TELL 0 ST CT
END

TO ERSH :n
MAKE "GEGEVENS [0 0 0 0 0 0 0 0 0 0 0
0 0 0 0]
IF EMPTYP :n [ERN "GEGEVENS STOP]
IF LISTP :n [PUTSH FIRST :n :GEGEVENS ER
SH BF :n] [PUTSH :n :GEGEVENS ERN "GEGEV
ENS]
END

TO ALLSH
OP [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
END

TO REM :opmerking
END

TO DOWNSH :n
PUTSH :n ( SE 0 BL GETSH :n
END

TO UPSH :n
PUTSH :n ( SE BF GETSH :n 0
END
```

```
TO PRSH :n
PR GETSH :n
END

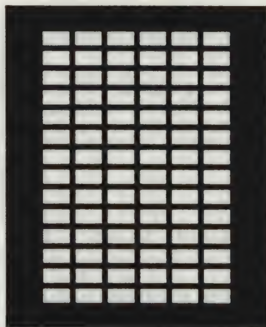
TO COPYSH :n1 :n2
PUTSH :n2 GETSH :n1
END

TO MOVESH :snelheid :n
IF [] = "n [STOP]
SETSH FIRST :n
WAIT :snelheid
PU FD 1
MOVESH :snelheid BF :n
END

TO ALL
OP [0 1 2 3]
END
```

Uitleg van de hulpprogramma's

Om de uitleg van de hulpprogramma's te illustreren maken we gebruik van de volgende sprook:



Als we nu PR GETSH 1 toetsen krijgen we de gegevens van dit sprook.

```
PR GETSH 1
255 129 129 129 129 129 129 129 129 129 129 129 129
129 129 255
```

Met TELL 0 SETSH 1 laten we Turtle 0 deze vorm aannemen.

Dan kijken we nu naar de hulpprogramma's.

SYSRESET : is een opdracht die u gebruikt om alles weer te laten zijn zoals het was toen u Logo startte. Alle Turtles krijgen weer hun eigen vorm, kleur en formaat, alleen Turtle 0 is zichtbaar, het beeldscherm en de tekenpennen hebben weer hun oorspronkelijke kleur.

- Met TELL 0 SETSH 1 SETC 120 SETSP 40 geven we Turtle 0 een vorm, een kleur en een snelheid. Om nu terug te gaan naar de 'oorspronkelijke' toestand toetsen we SYSRESET. Met SYSRESET hoeven we de vorm, de kleur en de snelheid niet apart te 'herstellen' maar gebeurt dit tegelijk.

ERSH : met deze opdracht 'wist' u alle gegevens van de genoemde vormen zonder de overige gebruikte variabelen te wissen.

- Toetsen we ERSH 1 (en RETURN) dan wissen we alle gegevens van sprook 1. Kijk maar:

```
PR GETSH 1
255 129 129 129 129 129 129 129 129 129 129 129 129
129 129 255

ERSH 1

PR GETSH 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

ALLSH : is geen opdracht, maar geeft als resultaat: alle vormen.

- Toetsen we ERSH ALLSH, dan wissen we de gegevens van alle sproken.

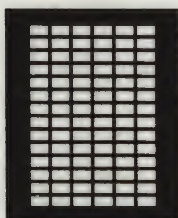
REM : soms is het, en dat geldt niet alleen bij procedures met sproken, handig om commentaar, opmerkingen enzovoort in een procedure te zetten. BASIC kent een REM-opdracht, dus waarom zou Logo daar niet over kunnen beschikken? Zoals u ziet is REM een lege procedure.

- Als voorbeeld de procedure die op de volgende bladzijde is afgedrukt.

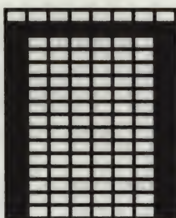
```
EDIT "VOORBEELDJE
TELL [0 1] ST SETSH 1
REM [Turtle 0 en 1 nemen hier de
vorm van sprook 1 aan]
RT 90 SETSP 100
REM [Beide Turtles worden hier 90
graden naar rechts gedraaid]
REM [en krijgen een snelheid van
100 eenheden]
REM []
REM [Zo zijn er nog veel meer]
REM [regels te maken.]
REM [Het zijn regels die door Logo]
REM [worden genegeerd.]
END
```

DOWNSH : is een opdracht die u kunt gebruiken als bijvoorbeeld blijkt dat een door u ontworpen vorm te groot is, of wanneer u iets bovenop de vorm wilt plaatsen. Met DOWNSH namelijk gaat genoemde vorm een rij (zie Shape Editor) naar beneden. De bovenste rij wordt dus een lege rij.

- Toetsen we EDSH 1 dan zien we:

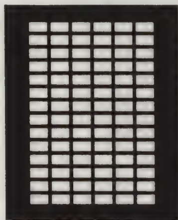


Met DOWNSH 1 en EDSH 1 krijgen we het volgende resultaat:

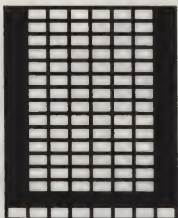


UPSH : is het tegenovergestelde van DOWNSH. Bij UPSH wordt de onderste rij van genoemde vorm een lege rij, en 'schuiven' alle rijen omhoog.

• We maken weer gebruik van sprook 1, zoals we dit aan het begin maakten. EDSH 1 geeft dan:



Met UPSH 1 en EDSH 1 krijgen we echter:



PRSH : laat u alle gegevens zien van de genoemde vorm.

• Uitgaande van de met UPSH 1 gemaakte sprook krijgen we het volgende:

```
PRSH 1
129 129 129 129 129 129 129 129 129 129 129 129 129
129 255 0
```

COPYSH : is een handige opdracht. Met deze instructie 'kopieert' u een vorm. Dit is bijzonder handig als u bijvoorbeeld met animatie bezig bent. Zoals u gelezen hebt, zullen bij een animatie de gebruikte sproken slechts weinig van elkaar verschillen. Als u nu eerst, laten we zeggen, drie gelijke vormen maakt, kunt u daarna de voor animatie noodzakelijke kleine verschillen aanbrengen. U zult merken dat COPYSH u veel werk scheelt.

• Met COPYSH 1 2 maakt u sprook 2 gelijk aan sprook 1.

MOVESH : geen volledige animatie-opdracht, maar net genoeg om u tijdens de ontwerpfase te laten zien wat het gevolg is van animatie van de genoemde vormen.
MOVESH vraagt twee invoergegevens:
- de tijd tussen het afbeelden van de verschillende vormen;

- de vormen (geplaatst in een lijst!) die moeten worden afgebeeld.

Opmerking. De procedure is niet recursief!

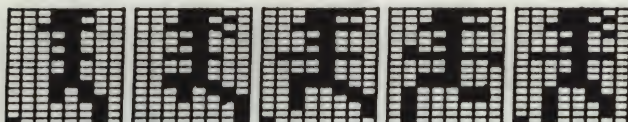
- Indien u de sproken 7, 8 en 9 ontworpen hebt en deze in een snelle animatie wilt zien, dan toetst u bijvoorbeeld:

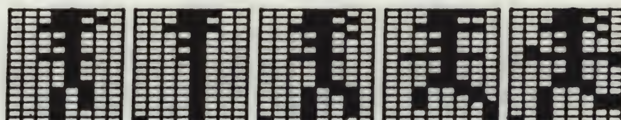
```
REPEAT 50 [MOVESH 2 [7 8 9]]
```

ALL : is geen opdracht, maar geeft als resultaat alle vier de Turtles. Eenvoudig aan te passen als u met een Logo werkt waarbij meer dan vier Turtles tegelijk 'actief' kunnen zijn.

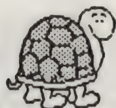
- Met TELL ALL SETSP 0 zetten we bijvoorbeeld alle Turtles stil. Met TELL ALL REPEAT 3 [FD 100 RT 120] laten we alle Turtles een driehoek tekenen.

Dat waren de hulpprocedures. Als u enige tijd met sproken hebt gewerkt, zult u snel in staat zijn uw eigen opdrachten te maken. Dat is een van de zeer aantrekkelijke kanten van Logo.

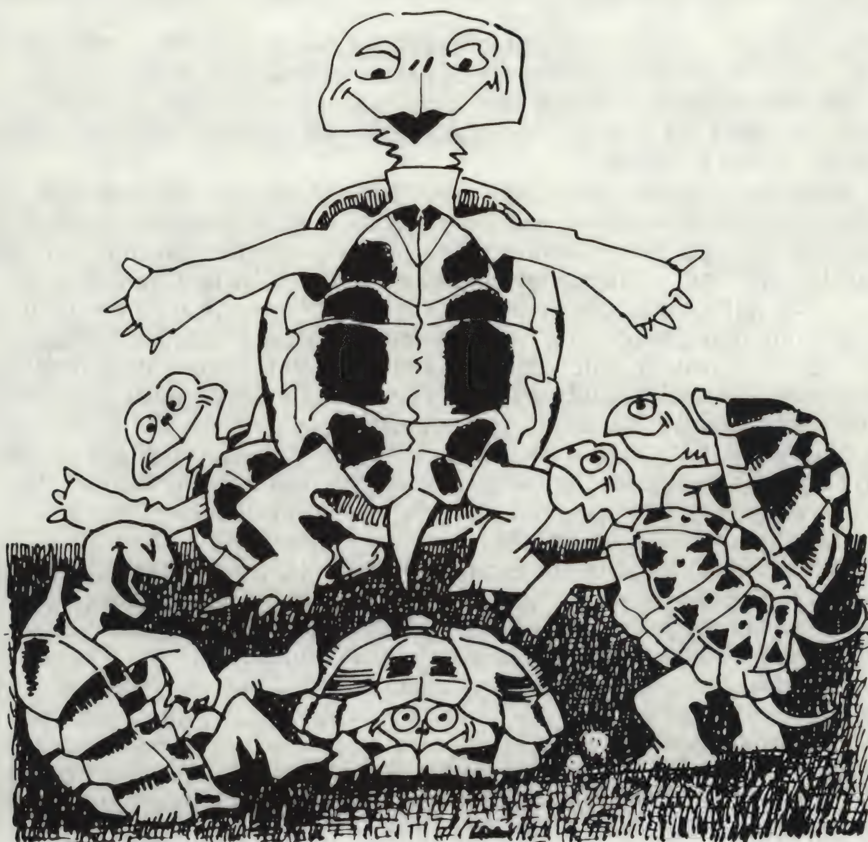




SPROKEN ONTWERP BLAD



Hoofdstuk 7 – ***LOGO MET KINDEREN***



7.1 Inleiding

Hoewel de programmeertaal Logo aanvankelijk ontwikkeld is ten behoeve van kinderen, is het beslist geen kindertaaltje. Dat heeft u hopelijk in de voorgaande hoofdstukken reeds kunnen constateren. Dit neemt echter niet weg dat Logo uitermate geschikt is voor kinderen. Natuurlijk thuis, maar ook op de scholen voor basisonderwijs en speciaal onderwijs. We kunnen Logo beschouwen als alternatief voor het, nu reeds, 'traditionele onderwijs met de computer'.

Er is namelijk een vrij grote kans dat het traditionele gebruik van computers in scholen - het Computer Ondersteund Onderwijs (C.O.O.) - niet meer zal blijken te zijn dan 'oude wijn in nieuwe zakken'. Zorgverbreding en kindgericht onderwijs zijn de magische woorden. In de meeste gevallen echter blijkt de huidige generatie computers niet eens zo veel meer te bieden dan de traditionele leermiddelen. Wellicht maakt het schoolbord plaats voor een beeldscherm en een leerboek voor een 'leerprogramma', maar in wezen verandert er niets.

O, zeker ... een computer boeit kinderen meer dan een leerboek, maar daardoor verandert het leerproces niet of nauwelijks. Al dan niet elektronisch zal het kind zich kennis eigen moeten maken. Kennis door anderen uitgezocht. We spreken hierbij van een 'produktiviteitsprincipe'.

Seymour Papert, geestelijk vader van Logo en leerling van Jean Piaget, bedoelde Logo als een alternatief voor bovengenoemde elektronische les- en overhoormachine. Logo is een alternatief voor de computer die 'het kind programmeert'. Immers in bovengenoemd produktiviteitsprincipe wordt het kind een 'van tevoren bepaalde' hoeveelheid kennis op een 'van tevoren bepaalde' manier aangeboden. Dat kan nuttig zijn voor het aanleren van bepaalde schoolse zaken (bijvoorbeeld spellingsregels), maar in feite voegt de computer niets nieuws toe aan het onderwijs.

Logo draait de zaken om. In een Logo-leeromgeving programmeert niet de computer het kind, maar het kind de computer. In een Logo-leeromgeving is het niet de computer die zegt wat er gebeuren moet, maar neemt het kind de initiatieven. Door de opdrachten die het kind de computer geeft kan het kind tekeningen maken, taalspelen maken, animaties en sproken ontwerpen, enzovoort. De Logo-computer is voor het kind desgewenst een tekenbord, een muziekinstrument of een typemachine.

Doordat het kind de computer leert, leert het zelf ook. Het kind leert, doordat het met de computer communiceert, analytisch en structureel te denken. Het kind leert problemen te formuleren en op te lossen. Het kind leert problemen te splitsen in kleinere problemen - Papert noemt dit 'denkklare brokken' - om zo tot de oplossing van grotere problemen te komen.

Tevens is een kind, terwijl het programmeert, bijna ongemerkt bezig met vele 'schoolse vaardigheden' zoals rekenen, wiskunde,

schrijven, manipuleren van informatie, ruimtelijke oriëntatie en nog veel meer. Ook kan ieder kind op zijn eigen niveau en in zijn eigen tempo met de computer aan de slag. In tegenstelling met de les-en-overhoor-computer blijft de Logo-computer eindeloos boeien en fascineren. Dat weet ik uit ervaring en daar wil ik u graag iets meer over vertellen.

7.2 Een praktijkverslag

In het najaar van 1983 zocht ik, aangestoken door de 'computerkoorts' naar een manier om met de computer op school aan de slag te gaan. Ik was toen als leerkracht werkzaam aan een school voor Speciaal en Voortgezet Speciaal Onderwijs. Om de in de inleiding genoemde redenen koos ik ervoor in mijn situatie de computer niet als 'les-en-overhoormachine' te gebruiken. Via Forth en de auteurstaal Pilot kwam ik toen bij Logo terecht.

Alle begin is moeilijk, de Logo-module (voor de Atari 8-bits computer) was duur en informatie over Logo was haast niet te krijgen. Door een artikel in een tijdschrift kwam ik echter op het spoor van het Logo Centrum Nederland in Nijmegen. De mensen daar maakten mij enigszins wegwijs in de wereld van Logo. Geen artikel of boek dat ook maar iets met Logo te maken had bleef ongelezen.

Intussen was ik thuis reeds druk aan het experimenteren met mijn pas gekochte Logo-module.

Toen ik Logo op school wilde introduceren, stuitte ik op een groot probleem: de Logo die ik bezat was Engelstalig. Op zich is dat niet zo bezwaarlijk natuurlijk, maar in mijn situatie lag dat wat moeilijker. Zoals zo vaak zou blijken lag de oplossing van dit probleem in de taal zelf. Logo is namelijk een proceduretaal en is als zodanig in staat nieuwe woorden te leren die zich net zo gedragen als de oorspronkelijke Logo-opdrachten.

Meteen ging ik aan de slag en het lukte! Om de opdracht FORWARD te vertalen gebruikte ik de volgende procedure:

```
EDIT "V :stap
FD :stap * 10
END
```

En dat was alles. In plaats van de Turtle met FORWARD (of FD) vooruit te sturen kon dat nu ook met V. De stapgrootte vermenigvuldigde ik met 10 om het werken met grote getallen te voorkomen. FD 90 kon nu dus ook worden bereikt met V 9. Op dezelfde manier heb ik toen meteen BACK (BK) vertaald in A.

Daarna waren RIGHT (RT) en LEFT (LT) aan de beurt, omdat dit de belangrijkste Turtle Graphics opdrachten zijn. Eigenlijk wilde ik hierbij het werken met graden omzeilen. Er moest een eenvoudige

manier zijn om weliswaar verschillende hoeken te kunnen kiezen, maar dan zonder het gebruik van graden. Een begrip als 180 graden rechtsom of 60 graden linksom zouden de kinderen nooit begrijpen. Ik heb toen gekozen voor drie verschillende procedures voor RT en drie procedures voor LT.

De procedures voor een 'kleine' bocht (30°) noemde ik R en L, voor een iets grotere bocht (45°) RR en LL en voor een grote bocht (90°) RRR en LLL. Door middel van combinaties zijn vele soorten hoeken te maken:
- L L voor een hoek van 60 graden naar links,
- RR R voor een hoek van 75 graden naar rechts,
- L RR voor een hoek van 15 graden naar rechts, enzovoort.
Natuurlijk moest ik hier concessies doen, maar Logo werd op deze manier en in deze situatie veel kindvriendelijker.

Zo zagen bijvoorbeeld de procedures voor de rechtsom-opdrachten eruit:

```
EDIT "R      EDIT "RR      EDIT "RRR
RT 30        RT 45         RT 90
END          END           END
```

Op deze manier heb ik veel LCSI Logo-primitieven vertaald. Ik zeg uitdrukkelijk VEEL, maar niet alle primitieven. Logo-primitieven die niet relevant waren voor mijn situatie heb ik buiten beschouwing gelaten. Onder de opdrachten die ik niet vertaalde waren functies zoals: POS, BG, PC, XPOS, YPOS, HEADING, WHO, NAMEP, EMPTYP, NUMBERP, maar ook opdrachten als: POALL, PODS, PONS, POPS, POTS, RECYCLE, NODES, enzovoort zijn door mij niet vertaald.

Toen ik klaar was met vertalen stuitte ik op een ander probleem. Door alle vertaalprocedures had ik namelijk zoveel geheugenruimte verbruikt, dat er nauwelijks nog ruimte was om te programmeren. Als oplossing koos ik voor losse vertaalde Logo-werelden, dat wil zeggen:
1. de Schildpadwereld (uitgebreid) vertaald;
2. Taal, Muziek en Sproken tezamen vertaald.
Zo verkreeg ik twee aparte vertalingen. Een voor de uitgebreide Schildpadwereld en een voor de gecombineerde Taal-, Muziek- en Sprokenwereld.

Ik was van mening dat de eerste Logo-wereld (Schildpadwereld) vrij uitgebreid moest zijn. Er moest, in het Nederlands, ook de mogelijkheid zijn voor eenvoudige sprookvormen, voor eenvoudige taalopdrachten en voor een eenvoudige geluidsopdracht.

Een moeilijkheid was nog het kiezen van de goede Nederlandse woorden voor de Logo-opdrachten. Nederlandse Logo-woorden moesten duidelijk, eenvoudig, kort (voor veelgebruikte opdrachten) en verantwoord zijn. De eerlijkheid gebiedt mij te zeggen, dat dit niet in één keer slaagde. Maar al te vaak bleek in de praktijk, dat de Nederlandse Logo-woorden te lang, te onduidelijk of te onlogisch waren. Maar al te vaak heb ik de Nederlandse Logo-woorden moeten

aanpassen. Arme kinderen, te vaak moesten zij wennen aan een nieuw Logo-woord.

Toch is de door mij gebruikte 'NederLogo' steeds duidelijker en gestructureerder geworden. Ik zal u de volledige lijst van NederLogo (Schildpadwereld) geven, maar natuurlijk staat het iedereen vrij de Nederlandse woorden naar zijn eigen smaak en behoefte aan te passen. Hoe dat moet zal iedereen nu wel bekend zijn.

Zijn de voornaamste Logo-woorden vertaald, dan gaat er een wereld voor de kinderen open: de wereld van de Logo-schildpad. Dan kan er naar hartelust worden geëxperimenteerd met lijnen, bogen, hoeken, kleuren, vormen en figuren. Sneller dan we denken komen kinderen dan vaak tot de meest fantastische resultaten.

7.3 NederLogo 2.7 - De complete lijst

NederLogo-opdrachten	LCSI-opdracht
V :n	FD :n
A :n	BK :n
R, RR en RRR	RT 30, 45 en 90
L, LL en LLL	LT 30, 45 en 90
HH :n	REPEAT :n
LEER "n	EDIT "n
MAAK "n m	MAKE "n m
K (KOM)	ST
W (WEG)	HT
S (SCHOON)	CS
WT (WEG TEKENING)	CLEAN
WW (WEG WOORDEN)	CT
ZEG :n	TELL :n
DOE :n	RUN :n
POP (PEN OP)	PU
PNR (PEN NEER)	PD
GUM	PE
GUM.	PX
KP :n (KIES PEN)	SETPN :n
KPK :n :n (KIES PEN KLEUR)	SETPC :n :n
KBK :n (KIES BEELD KLEUR)	SETBG :n
KSK :n (KIES SCHILDPAD KLEUR)	SETC :n
ZETVAART :n	SETSP :n
LT (LEES TEKEN)	RC
LL (LEES LIJST)	RL

NederLogo-opdrachten	LCSI-opdracht
PAKWEG	RANDOM
DRUK	PRINT
TOON :n :n :n :n	TOOT :n :n :n :n
MV :n (MAAK VORM)	EDSH :n
GV :n (GEEF VORM)	GETSH :n
KV :n (KRIJG VORM)	SETSH :n

7.4 Onderwijsprogramma's met Logo

Mocht uit het voorafgaande de indruk gewekt zijn dat de kinderen de Logo-computer alleen te zien krijgen als een soort elektronisch tekenblok, dan is dit niet juist. Juist om ervoor te zorgen dat de Logo-computer blijft boeien is het zaak kinderen ook andere dingen aan te bieden. Hierbij moet worden opgemerkt, dat de kinderen waar ik destijds les aan gaf nog niet toe waren aan het schrijven van taal-, muziek- of sprokenprogramma's. Ik liet de kinderen dan ook werken met door mij gemaakte of bewerkte Logo-programma's (deze gebruiken de kinderen dus naast hun eigen programmeeractiviteiten). Vaak vroegen de kinderen erom te mogen werken met deze programma's.

Deze programma's werden door mij gebruikt:

1. als introductie
2. als spel
3. ondersteunend en remediërend.

1. Logo-programma's als introductie

De kinderen waar ik mee werkte hadden aanvankelijk nogal wat problemen met het toetsenbord van de computer. Enkele van de voornaamste problemen waren de repeterende toetsen en de letterherkenning. Ook 'veranderende dimensie' (op het televisiescherm wordt naar voren: naar boven) en de 'Turtle Geometry' (richtingsbepaling vanuit de Turtle) waren zaken die aanvankelijk problemen gaven. Voor verschillende kinderen en op verschillende niveaus zijn toen programmaatjes geschreven.

Hier waren onder andere programma's bij voor het:

- tekenen met de Turtle zonder gebruik van het toetsenbord;
- tekenen met enkele letters (V,A,L,R) of zelfs alleen met de cursortoetsen;
- verplaatsen van de Turtle naar een bepaalde plaats (gebruikmakend van de Joystick).

Op velerlei manieren en met vaak maar heel korte programmaatjes is *Logo aan het kind aan te passen en hoeft het niet andersom.*

2. Logo-programma's als spel

Het idee van de computer als spelletjesmachine is moeilijk te bestrijden. Steeds weer vroegen de kinderen om een Logo-spel. En omdat het spreekwoord luidt: 'If you can't beat'em, join'em', heb ik toen - met de kinderen - verschillende spelen gemaakt. Voor veel ondersteunende en remediërende programma's is trouwens ook een spelvorm gekozen.

3. Ondersteunend en remediërend

Het zal iedereen inmiddels duidelijk zijn dat Logo niet bedoeld is voor C.O.O. (Computer Ondersteund Onderwijs). Voor 'les-en-overhoor'-programma's (Drill and Practice) is Logo nooit bedoeld geweest. Toch kan het soms verhelderend en stimulerend werken om met kinderen samen een C.O.O.-programma te schrijven. Want wat is er nu beter, wanneer een kind problemen heeft met bijvoorbeeld werkwoordsvervoegingen, dan het kind daar zelf een programma over te laten schrijven? Op deze manier is het kind op een speelse wijze bezig met een voor hem/haar essentieel (te remediëren) onderwerp.

Het is overigens wel aan te bevelen om voor dit doel met zogenaamde 'raamprogramma's' te werken. Een raamprogramma is dan een kant-en-klaar programma, dat op eenvoudige wijze kan worden aangepast. In een raamprogramma hoeft slechts de informatie (werkwoorden, sommen, enzovoort) te worden gewijzigd, zodat niet steeds een nieuw programma hoeft te worden geschreven.

Het is daarbij verstandig de informatie in een lijst te zetten, waarvan ieder element zowel de opgave als het antwoord bevat. Een paar voorbeelden:

```
MAKE "sommen [[[3+2=] 5] [[4+4=] 8] [[1+3=] 4]]
MAKE "werkwoorden [[fietst fietste] [wandelt wandelde]
  [lacht lachte]]
MAKE "reken [[[2.4=6] +] [[3.1=2] -] [[4.4=8] +]]
```

U ziet het, zowel de opgaven als de antwoorden staan in een lijst.

In een raamprogramma moeten nu de volgende procedures voorkomen:

- een procedure die de informatie bevat (sommen, werkwoorden, enzovoort), vastgelegd in een lijst;
- een procedure die uit de hierboven genoemde lijst willekeurig één element haalt;
- een procedure die de opgave afdrukt;
- een procedure die het gegeven antwoord vergelijkt met het juiste antwoord;
- een procedure voor de goede antwoorden;
- een procedure voor de foute antwoorden;

- een slotprocedure waar bijvoorbeeld het aantal goede/foute antwoorden wordt vermeld.

Als voorbeeld van zo'n raamprocedure ziet u hieronder SOMMENMAKEN. SOMMENMAKEN bestaat uit tien procedures:

- SOMMENMAKEN : de hoofdprocedure; hier wordt de vraag gesteld hoeveel opgaven er gegeven moeten worden. Ook wordt het aantal goede en foute antwoorden op 0 gezet.
- RIJTJE : de procedure die ervoor zorgt dat het opgegeven aantal opgaven één voor één en onder elkaar wordt afgedrukt.
- SOMMEN : de procedure waar de opgaven en antwoorden in staan. Door MAKE "som KIES :sommen wordt uit de lijst opgaven/antwoorden willekeurig één element gekozen.
- BEELD : de procedure die het eerste gedeelte van :som (de opgave) afdrukt en wacht op het in te typen antwoord. Daarna wordt het ingetypte antwoord vergeleken met het laatste gedeelte van :som (waar het antwoord staat).
- GOED : de procedure die in werking treedt als er een goed antwoord wordt gegeven. Het aantal goede antwoorden wordt dan met één opgehoogd.
- FOUT : de procedure die in werking treedt als er een fout antwoord wordt gegeven. Het aantal foute antwoorden wordt dan met één opgehoogd.
- KIES : de procedure die ervoor zorgt dat op willekeurige wijze een opgave (en een antwoord) wordt gekozen.
- ITEM : de procedure die voor KIES nodig is.
- RN : de procedure die als het ware het antwoord 'leest'.
- SLOT : de procedure die het raamprogramma besluit. Hier wordt aangegeven hoeveel opgaven er zijn gemaakt en hoeveel goede of foute antwoorden er waren. Tevens stelt SLOT de vraag of het programma nogmaals moet draaien.

Hier volgen de procedures:

```
TO SOMMENMAKEN
CT ERNS
MAKE "aantalgoede 0
MAKE "aantalfoute 0
MAKE "R 2
TYPE [Hoeveel sommen wil je maken ?]
MAKE "aantal RN
RIJTJE :aantal
END
```

```
TO RIJTJE :N
IF :N = "O [SLOT STOP]
SOMMEN
SETCURSOR LIST 14 :R
BEELD
MAKE "R :R + 1
RIJTJE :N - 1
END

TO SOMMEN
MAKE "sommen [[[3 + 4 =] 7][[2 + 1 =] 3]
[[5 + 2 =] 7] [[4 + 1 =] 5][[2 + 2 =] 4]]
MAKE "som KIES :sommen
END

TO BEELD
TYPE FIRST :som
MAKE "antwoord RN
IF EQUALP LAST :som :antwoord [GOED] [FOUT]
END

TO GOED
MAKE "aantalgoede :aantalgoede + 1
END

TO FOUT
MAKE "aantalfoute :aantalfoute + 1
END

TO KIES :N
OP ITEM ( 1 + RANDOM COUNT :N ) :N
END

TO ITEM :N :O
IF EMPTYP :O [OP "]
IF :N = 1 [OP FIRST :O]
OP ITEM :N - 1 BF :O
END

TO RN
OP FIRST RL
END

TO SLOT
PR ( SE [Je hebt nu] :aantal "sommen "gemaakt.
PR ( SE [Daarvan waren er :] :aantalgoede "goed.
PR ( SE [Daarvan waren er :] :aantalfoute "fout.
PR []
PR [Nog een keer J / N ?]
MAKE "keuze RC
IF :keuze = "J [SOMMENMAKEN] [CT STOP]
END
```

Een ander programma dat ook grafisch aardig is om te bekijken is: KLOKKIJKEN. In dit programma wordt op het beeldscherm een cijferloze klok getekend, waarna de wijzers een (op RANDOM-wijze bepaalde) tijd aangeven. Onder op het scherm verschijnt dan:

Hoe laat is het nu? (bij de hele uren)
Het is nu half: (bij de halve uren)

Wanneer dan de juiste tijd/het juiste uur wordt ingetypt, verschijnt er bijvoorbeeld:

Jazeker
Het is inderdaad . uur

Na tien opgaven wordt het aantal goed en fout gemaakte opgaven vermeld.

Dit programma kan eenvoudig worden gewijzigd in bijvoorbeeld klokkijken met kwartieren, of in digitaal klokkijken. Dan moeten er in de volgende procedures verschillende dingen veranderd worden:

1. de opgaven in KIESUUR (voor digitaal klokkijken zou een element er zo uit kunnen zien: [90 210 7.15], waarbij 90 de richting van de grote wijzer, 210 de richting van de kleine wijzer en 7.15 de digitale tijd is);
2. de tekst in KIJKNA;
3. de teksten in PRIMA en FOUT;
4. tenslotte ook de tekst van SLOT.

Hier is het programma KLOKKIJKEN:

```
TO KLOK
MAKE "g 0 MAKE "f 0
MAKE "GOED [Prima Keurig Jawel
 Inderdaad Jazeker Voortreffelijk]
MAKE "FOUT [Jammer Fout Onjuist
 Nee.... Ojee!!]
TELL 0 HT FS VK 60 VK 63 VK 66 VK 70
TELL [1 2] WIJZERS
SS
ZETWIJZERS 10
END

TO ITEM :N :O
IF EMPTYP :O [OP "]
IF :N = 1 [OP FIRST :O]
OP ITEM :N - 1 BF :O
END
```

→

```
TO KIES :TIJDEN
OP ITEM ( 1 + RANDOM COUNT :TIJDEN ) :TIJDEN
END

TO WIJZERS
PU SETPOS [0 20] PD
REPEAT 12 [PU FD 52 PD FD 3 PU BK 55 LT 30]
PU SETPOS [0 20]
END

TO VK :n
PU SETPOS [0 20] FD :n RT 90 PD
REPEAT 4 [FD :n RT 90 FD :n]
END

TO HUIS
TELL [1 2] PE
SETPOS [0 20]
PD
END

TO KLEINEWIJZER
TELL 1 HT SETPOS [0 20] SETH FIRST BF :TIJD PD FD 25
END

TO GROTEWIJZER
TELL 2 PU HT SETPOS [0 20] PD
SETH FIRST :TIJD
FD 45
END

TO KIESUUR
MAKE "TIJDEN [[0 30 1][0 60 2][0 90 3][0 20 4]
 [0 150 5][0 180 6][0 210 7][0 240 8][0 270 9]
 [0 300 10][0 330 11][0 360 12][180 30 1][180
 60 2][180 90 3][180 120 4][180 150 5][180
 180 6][180 210 7] [180 240 8][180 270 9]
[180 300 10][180 330 11][180 360 12]]
MAKE "TIJD KIES :TIJDEN
END

TO ZETWIJZERS :N
IF :N = "0 [SLOT STOP]
ERN "TIJD
KIESUUR
GROTEWIJZER
KLEINEWIJZER
KIJKNA
WAIT 50 CT HUIS
ZETWIJZERS :N - 1
END
```

→

```
TO FOUT
MAKE "f :f + 1
MAKE "NEE KIES :FOUT
PR :NEE
IF EQUALP FIRST :TIJD "O [PR ( SE [Het was]
 LAST :TIJD "uur][PR (SE [Het was half]
 LAST :TIJD
END

TO PRIMA
MAKE "g :g + 1
MAKE "JA KIES :GOED
PR :JA
IF EQUALP FIRST :TIJD "O [PR ( SE [Het is
 inderdaad] LAST :TIJD "uur][PR ( SE [Het is
 inderdaad half] LAST :TIJD
END

TO KIJKNA
IF EQUALP FIRST :TIJD "180 [TYPE [Het is nu
 half :]][PR [Hoe laat is het nu ?]]
MAKE "G.A LAST :TIJD
MAKE "A FIRST RL
IF EQUALP :A :G.A [PRIMA][FOUT]
END

TO SLOT
TS CT
REPEAT 10 [PR []]
PR [Dat waren de hele en de halve uren.]
PR []
PR ( SE [Je had] :g "opdrachten "goed
PR ( SE [Je had] :f "opdrachten "fout
PR []
PR [Nog eens ????]
PR [Type dan : KLOK]
END
```

Voor deze en andere in Logo geschreven programma's is gebruik gemaakt van de vele mogelijkheden die Logo als taal bezit. Op deze manier zijn verschillende schoolse zaken aan de orde geweest:

- tellen
- het omgaan met hoeveelheden
- lezen / taalbegrip
- ruimtelijke begrippen
- ruimtelijke oriëntatie
- visuele / auditieve waarneming
- klokkijken

enzovoort, enzovoort.

Vanzelfsprekend komen vele van deze leer/vormingsgebieden ook aan de orde wanneer Logo als 'leeromgeving' wordt gehanteerd. Hier zal de nadruk dan ook op moeten blijven liggen. Maar omdat het zo eenvoudig is remediërende programma's te maken en omdat de kinderen daardoor de Logo-computer ook eens van een andere kant zien, leek het goed u dit niet te onthouden.

7.5 Logo in de klas

De veranderingen die Seymour Papert in het onderwijs wil aanbrengen zullen waarschijnlijk niet op korte termijn worden gerealiseerd. Maar we kunnen Logo natuurlijk ook binnen de huidige onderwijsstructuur hanteren. Daarbij moet Logo niet fungeren als doel, maar als middel, als didactisch hulpmiddel, dus Logo in de klas als 'leeromgeving'; Logo als 'omgeving' waarin het kind gestructureerd en probleemoplossend leert denken.

Ook in onze dagelijkse onderwijspraktijk is met Logo een omgeving te scheppen waarin het kind op een speelse wijze bezig is 'zichzelf' te ontplooien. In een Logo-leeromgeving is een kind op geheel eigen wijze bezig met allerlei leer/vormingsgebieden die elders 'apart' op het lesrooster staan. In een Logo-leeromgeving is het kind bezig met visuele/auditieve perceptie (zintuiglijke waarneming via ogen en oren), taal/spelling, ruimtelijke oriëntatie en begrippen, lateralisatie (links/rechts begrip), rekenen, omgaan met verhoudingen, enzovoort.

Maar het allerbelangrijkste voor het kind is voor de leerkrachten vaak het moeilijkste: in de Logo-leeromgeving neemt het KIND het initiatief. Niet de computer zegt wat er moet gebeuren, niet de leerkracht, maar het KIND deelt hier de lakens uit. Op geheel eigen wijze ontdekt en onderzoekt het KIND de Logo-wereld. Wellicht is dit de reden waarom verschillende leerkrachten sceptisch tegenover Logo staan. Zou het dan zo moeilijk zijn te accepteren dat een kind ook zijn eigen leraar kan zijn?

Een voorbeeldje. Een kind is met de Logo-Turtle aan het tekenen en wil graag een cirkel maken. Het kind doet verwoede pogingen om de Turtle een cirkel te laten tekenen:

```
REPEAT 10 [FD 10 RT 60]
REPEAT 100 [FD 5 RT 30]
```

maar het wil maar niet lukken. Hoe groot is dan de verleiding voor de leerkracht om

```
REPEAT 36 [FD 10 RT 10]
```

in te toetsen? Uiteindelijk, via vierkanten, vijfhoeken, zeshoeken, enzovoort, zal het kind er uit zichzelf wel achter komen. Dat leerproces is, volgens de Logo-filosofie, veel belangrijker dan het resultaat: de cirkel.

Als de leerkracht het kind vertelt hoe een cirkel getekend moet

worden, wordt dit proces van 'al doende leren' doorbroken. Het is daarom ook zaak van Logo geen 'vak' te maken waarvoor het kind een cijfer kan krijgen. Veeleer is Logo een vorm van wereldverkenning waarbinnen het kind spelenderwijs met vele schoolse vaardigheden bezig is.

Logo kan voor een kind een individuele activiteit zijn. Maar dat hoeft niet! Gebleken is dat kinderen elkaar juist kunnen stimuleren en motiveren.

Als materiaal bij het scheppen van een Logo-omgeving kunnen de oefenboekjes van het Logo Centrum Ede worden gehanteerd. Deze boekjes zijn door Willem Heyster samengesteld en bevatten zeer veel oefenmateriaal (het adres vindt u in appendix 5). Veel van dit materiaal is in dit boek als illustratie gebruikt.

Natuurlijk moeten we reëel blijven. Een Logo-leeromgeving kan in lang niet iedere schoolsituatie optimaal worden gecreëerd. Zelfs in het gunstigste geval, waarbij in iedere groep ten minste één computer staat, is nog de tijd dat ieder kind de mogelijkheid heeft ermee te werken vaak veel te kort.

Hierbij moet worden aangetekend dat het speciaal en voortgezet speciaal onderwijs een streepje voor hebben. Deze vormen van onderwijs hebben namelijk door hun groepsgrootte vaak meer differentiatiemogelijkheden dan de basisscholen. Maar in iedere situatie kan naar een compromis worden gezocht.

In mijn situatie wordt er wekelijks een volle dag aan Logo besteed. De kinderen werken dan individueel of in groepjes om beurten. Zo komt ieder kind aan bod en kan op zijn eigen niveau bezig zijn. Erg snel blijken namelijk al grote verschillen tussen de diverse kinderen te bestaan. Maar dat geeft niet. Iedereen kan in zijn eigen tempo en op geheel eigen wijze te werk gaan. Ieder kind heeft een eigen diskette waar de procedures, hulpprogramma's en eventueel spelen op staan waar het kind mee bezig is. De kinderen worden zo weinig mogelijk gestoord, maar weten dat ze te allen tijde een beroep op de leerkracht kunnen doen. Het gevaar van een dergelijke manier van werken is, dat je het zicht kunt verliezen op waarmee de kinderen bezig zijn. Hoe lost een kind de problemen op die zich voordoen? Hoe beheerst een kind de verschillende programmeervaardigheden? Van tijd tot tijd is het bijzonder interessant om de kinderen te observeren wanneer zij aan het werk zijn.

En omdat het kind de verschillende 'schoolse' vaardigheden hier in een 'ongedwongen en eigen' situatie hanteert, maak ik zo nu en dan ook enige aantekeningen (achteraf) over het inzicht van het kind in verhoudingen, het omgaan met hoeveelheden, lateralisatie (links/rechts begrip), rekenkundige bewerkingen, enzovoort. Juist in een situatie als deze is het buitengewoon boeiend te zien hoe kinderen de 'schoolse' vaardigheden hanteren. Hier ligt er immers geen dwang op, hier is het kind de situatie meester.

Aanvankelijk maakte ik losse aantekeningen van ieder kind. Later echter besloot ik dit met behulp van een soort Logo-Observatieformulier te doen. Geenszins om aan te geven hoe 'goed' een kind in Logo is, maar om te noteren hoe een kind 'schoolse' vaardigheden

in de 'eigen' situatie hanteert. Dit observatieformulier is slechts voor mijn eigen administratie. Ik wil het u echter niet onthouden. Het staat iedereen vrij een en ander aan te passen aan de eigen situatie. U vindt het op de volgende bladzijde.

Toelichting op het Logo-Observatieformulier

Dit formulier gaat uit van de computertaal Logo als leeromgeving voor het probleemoplossend denken (P.O.D.). Nu is de vaardigheid van het P.O.D. moeilijk meetbaar en niet duidelijk te omschrijven. Daarvoor is de volgende oplossing gekozen. Het formulier toont diverse vaardigheden aan de hand waarvan een beeld van het P.O.D. kan worden gevormd.

Het formulier gaat achtereenvolgens in op de volgende onderdelen:

- perceptie
- ruimtelijk inzicht
- rekenen
- hantering Logo

Elk van deze aspecten is weer uitgesplitst in diverse vaardigheden.

Het formulier beoogt geenszins een volledig beeld te geven van de beheerste vaardigheden, daar is het veel te beperkt voor. De bedoeling is een globale indruk te geven van de beheerste vaardigheden binnen de leer- en programmeeromgeving van Logo.

Binnen het 'probleemoplossend' denken zijn de volgende fasen te onderscheiden:

1. Waarneming/formulering van het probleem. Wat moet er gebeuren? Bijvoorbeeld: welke grafische voorstelling moet worden gemaakt?
2. Welke weg moet bewandeld worden om dit probleem op te lossen? Opsplitsing van het probleem in kleinere eenheden.
3. Welke opdrachten zijn noodzakelijk voor dit probleem? Nu en in een later stadium.
4. Welke subprocedures zijn noodzakelijk om dit probleem op te lossen?
5. Aanpak van het probleem:
   - gebruik van de juiste opdrachten
   - gebruik subprocedures
   - gebruik variabelen, recursie, enzovoort
   - schrijven van de procedures.
6. Het testen, foutvrij maken ('debuggen') en laten werken van de procedures.

Noodzakelijk voor deze manier van werken zijn de in het formulier genoemde en geteste vaardigheden, hoewel het een geen voorwaarde voor het ander hoeft te zijn. Met andere woorden: er kan reeds, met



NAAM : ....

> Onderdeel :   > Datum :

Perceptie :
....

* waarneming globaal :
* idem gedetailleerd :
* cijfer/letter her. :
* idem lezen :

Ruimtelijk inzicht :
....

* globaal :
* omgaan ruimt.beg. :
* verhoudingen :
* lateralisatie :
* hoeken :

Rekenen :
....

* globaal :
* omgaan hoeveelh. :
* omgaan cijfers :
* sequensen :
* rek.bewerkingen :
* kardineren :

Gebruik Logo :
....

* hantering joystick :
* gebruik toetsen :
* gebruik b.c VARL :
* edit functie :
* gebruik variabelen :
* gebruik recursie :
* gebruik algemeen :
* werkhouding :

vallen en opstaan, geprogrammeerd worden zonder kennis of programmeervaardigheden. *Trial and Error* heet dit in onderwijstermen, ofwel de benadering: al doende leren. Bij het toenemen van de kennis en de programmeervaardigheden zal ook de mate van P.O.D. toenemen. De basis is, zoals het in de Logo-wereld heet: leren zonder uitdrukkelijke instructie.

Door het programmeren (hoe eenvoudig ook) van een machine, de Logo-computer, leren kinderen probleemoplossend denken. En dat is voor de makers van Logo ooit eens de reden geweest om deze taal te ontwikkelen.

Soms kan het zinvol zijn te zien hoe een kind een bepaald Logo-probleem aanpakt. Het gaat dan duidelijk niet om het te maken produkt (een tekening), maar om de wijze waarop het kind het probleem benadert. Welke stappen onderneemt het kind om tot de oplossing van dat specifieke probleem te komen:

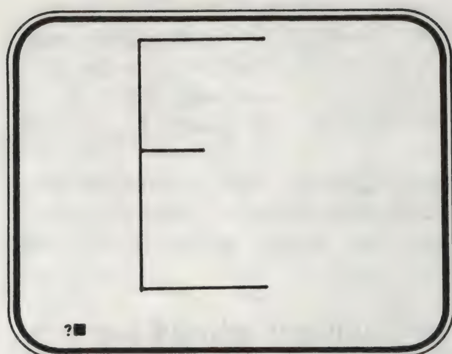
- Observeert het kind de te maken figuur goed?
- Weet het kind de opdrachten die het op dat moment kent juist te hanteren?
- Is het kind in staat op een juiste wijze verhoudingen/hoeken enzovoort te interpreteren?
- Hanteert het kind het probleem op een logische wijze?
- Hoe reageert het kind op fouten die het bij het programmeren maakt?
- Hoe is de interactie van het kind met zijn/haar Logo-computer?

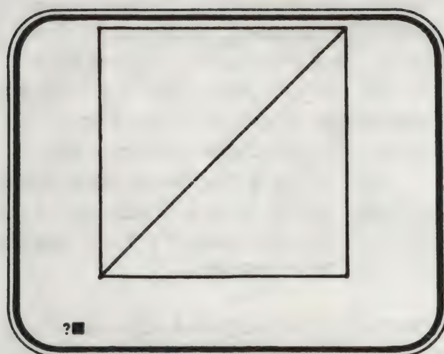
Deze vragen kunnen met observatie vaak niet worden beantwoord.

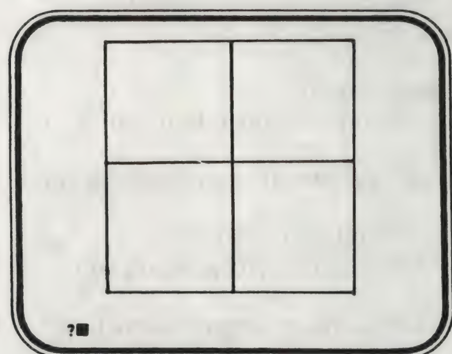
Toch vond ik het aardig eens te kijken op welke manier ieder kind in zijn/haar Logo-omgeving bezig was. Bij wijze van proef heb ik de kinderen toen een zelfde opdracht gegeven. Elk van deze kinderen, dat wist ik, zou geen enkel probleem hebben met het maken van de gevraagde figuren. Het ging er daarom ook niet om *of* de kinderen de figuren konden maken, maar *hoe* de kinderen te werk zouden gaan. Ook hierbij ben ik uitgegaan van het materiaal van het Logo Centrum Ede. Ter illustratie vindt u hier een voorbeeld van zo'n opdracht.

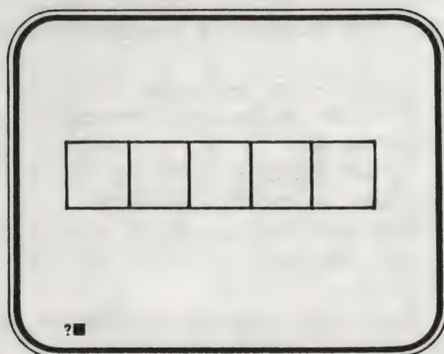
Opdracht: maak de volgende vier figuren

Opmerking. Bij het maken van deze figuren komen de diverse programmeervaardigheden overduidelijk aan de orde: het waarnemen van de figuur; gebruik van de juiste opdrachten; ruimtelijk inzicht; omgaan met hoeveelheden; cijfersymbolen; relaties; verhoudingen; manier van probleemoplossen, enzovoort. Bij al deze figuren wordt voornamelijk gekeken: hoe lost het kind het probleem op en welke programmeervaardigheden ontbreken nog?









7.6 *Afsluiting*

Het zal iedereen duidelijk zijn dat Logo, hoewel het geen kindertaaltje is, toch uitstekend aanslaat bij kinderen. Logo als programmeertaal, Logo als leeromgeving, thuis en op school. In Amerika is lange tijd sprake geweest van een ware Logo-manie. Logo is daar dan ook uitgeroepen tot de Educatieve Software van het jaar. En ook u zult, bij het doorlezen van dit boek, tot de ontdekking zijn gekomen dat de drempel om met Logo te beginnen wel erg laag is. Zo laag zelfs, dat zelfs heel jonge kinderen al met Logo bleken te kunnen werken.

Het mooiste hierbij is natuurlijk de Vloerschildpad (Floor Turtle), een robot-achtig apparaat dat, net als de beeldschermschildpad, reageert op de via het toetsenbord gegeven opdrachten. Helaas is de vloerschildpad nogal kostbaar en heeft lang niet iedere Logo-versie de mogelijkheden om ermee te werken.

Wel moet worden vastgesteld dat Logo, ook in het huidige onderwijssysteem, een volwassen plaats zou kunnen innemen. Het is

dan ook verheugend te zien dat steeds meer scholen de Logo-mogelijkheden van hun computer gaan inzien. Belangrijk in deze ontwikkeling is, dat er steeds meer Nederlandstalige Logo-versies verschijnen. De Nederlandse Logo's voor MSX en Commodore zijn hier een duidelijk voorbeeld van.

Een probleem echter is nog de achtergrondinformatie. De handleiding die bij het Logo-pakket geleverd wordt is vaak onvoldoende. Een gunstige uitzondering hierop vormen opnieuw de twee reeds eerder genoemde Nederlandstalige Logo's.

Het is duidelijk dat de belangstelling voor Logo toeneemt. Hierdoor zal ook de hoeveelheid literatuur over Logo toenemen. En dat is een goede zaak.

Hopelijk heeft u uit het voorafgaande een beeld kunnen krijgen van de mogelijkheden van Logo met kinderen, de mogelijkheden van Logo ook als leer- en programmeeromgeving op scholen. Indien mogelijk willen de verschillende Logo Centra u graag helpen als u vragen hebt. De adressen vindt u in appendix 5.

Hoofdstuk 8

LOGO ..... EEN TAAL; VELE DIALECTEN



8.1 Bespreking van de meestgebruikte Logo's

Het is jammer maar waar..., ook een taal als Logo is niet universeel. Op vele computers is te werken met Logo, maar er zijn steeds kleine verschillen. Eén taal met vele dialecten, het is hetzelfde probleem als met BASIC. In dit hoofdstuk zullen de bekendste Logo-versies besproken worden.

Tevens vindt u een zeer handige vergelijkingstabel, waar de meest gebruikte Logo-opdrachten in te vinden zijn voor de volgende machines: Apple, Atari 800/130, Atari 1040/520/260 ST, Acorn en BBC, Commodore, Sinclair Spectrum en de MSX-computers.

Op het moment zijn drie Nederlandstalige Logo-versies op de markt: een oude TI Logo, de Nederlandse Commodore-Logo en de Nederlandse MSX-Logo. Maar u kunt, zoals u heeft gezien, vrij eenvoudig uw eigen Nederlandstalige Logo maken.

Laten we eerst maar eens naar enkele Logo-versies gaan kijken.

Apple Logo

Voor de Apple-computers zijn er verschillende Logo-versies te verkrijgen. Zo zijn er de Terrapin Logo en de Krell Logo. Voor Apple is een Logo geschreven door het softwarebedrijf Logo Computer Systems Incorporated (LCSI), hetzelfde bedrijf dat ook de Logo's voor de Atari 800/130 en de Spectrum heeft ontwikkeld. Bovendien is er voor de Macintosh wel een heel speciale Logo verschenen: de ExperLogo.

Experlogo is een fantastische, maar vrij prijzige Logo-versie die behalve ongekende grafische mogelijkheden ook nog over een zeer uitgebreide lijstbewerking beschikt. Wat het grafische deel betreft: ExperLogo is bijzonder snel in het tekenen, vandaar dat de Turtle vervangen is door de Bunny (het konijn). Hoewel in ExperLogo de kleuren ontbreken, is het een hele belevenis ermee te kunnen werken.

In de tabel wordt echter uitgegaan van de LCSI Logo, omdat deze bij Apple de meest gebruikte is.

Atari Logo

Voor Atari zijn er twee Logo-versies beschikbaar: een voor de 8-bits modellen (600/800/130): de LCSI Logo, en een vóor de 16-bits machines (260/520/520+/1040): een Atari Logo, ontwikkeld in nauwe samenwerking met Digital Research (DR. Logo). In dit boek is al genoeg gezegd over de Atari LCSI Logo (daarmee heeft de auteur van dit boek lange tijd en naar volle tevredenheid gewerkt).

De DR.Logo wordt, een pluspunt voor Atari, stan-

daard bij de machine geleverd (op diskette). Het is een fantastische belevenis om met deze Logo te werken: deze Logo is namelijk volledig met GEM*) (dus met 'vensters' met de muis) te besturen vanuit diverse menu's. Zo kunnen we direct na het starten al kiezen uit een Desk File Run Edit en Settings menu. Om een voorbeeld te noemen: wanneer we met de muis het Edit-menu kiezen, kunnen we, eveneens zonder het toetsenbord aan te raken, de volgende opdrachten kiezen: Workspace, File, Save Edit, Abandon, Mark, Cut, Copy, Paste, Top, Bottom, Center, Page Up, Page Down, →Line en ←Line. Zo is de DR.Logo een zeer bijzondere Logo. Wel moet nog worden opgemerkt dat in deze Logo de muziek- en sprookmogelijkheden ontbreken, maar daar staat een zeer uitgebreide lijstbewerking tegenover.
In de tabel worden zowel de LCSI als deze DR. Logo besproken.

BBC Logo — De meest gebruikte Logo voor de BBC-computer wordt geleverd op ROM en moet in de machine worden ingebouwd. Deze Logo beschikt tevens over een losse cassette/diskette met de zogenaamde 'Extensions' compleet met vele voorbeeldprogramma's. Voor de BBC Logo is tevens veel randapparatuur te verkrijgen (veelal in Engeland) zoals: vloerschildpadden, buggies, robots en een zogenaamd 'concept keyboard'. Het was een genoegen er langere tijd mee te kunnen werken.
In de vergelijkingstabel vindt u de opdrachten van BBC's eigen Logo-versie (door Acornsoft ontwikkeld).
Voor de BBC is ook de professionele taal LISP te verkrijgen.

Commodore Logo — Voor de Commodore-computers zijn momenteel drie Logo-versies te verkrijgen. U heeft de keuze uit Engelstalige en Nederlandse Logo's. De Engelse Commodore Logo is een heel mooie met zeer veel mogelijkheden (vooral op het gebied van spraken en muziek). Deze 'Terrapin Logo' is, wat we zouden kunnen noemen, een verbeterde versie van de Apple Terrapin Logo.

---

*) GEM betekent: Graphics Environment Manager en is een grafisch georiënteerd programma dat tussen de gebruiker en de systeemopdrachten van de computer in staat. Met behulp van grafische symbolen op het scherm worden de diverse systeemopdrachten aangeduid. Deze pictogrammen (voor inhoudsopgave van disk, formatteren, bestanden kopiëren, enzovoort) worden op het beeldscherm met een pijltje aangewezen. Op deze manier zijn systeemopdrachten zeer snel aan de computer door te geven.

Zoals gezegd is dit een erg uitgebreide Logo-versie. Evenals bij de BBC het geval is, wordt ook deze Logo geleverd met een zogenaamde 'Utility Disk'. Op deze Utility Disk staan onder andere demonstratieprogramma's, hulpbestanden (als de Sprite Editor) en een zeer uitgebreid Logo-muziekprogramma.
Omdat voor Commodore momenteel een Nederlandse Logo beschikbaar is, wordt deze 'Terrapin Logo in de winkels voor een 'bodemprijs' verkocht.
Nieuw voor Commodore is de Nederlandse Logo (uitgever: Malmberg). Opvallend bij deze prachtige Logo is het uitgebreide en duidelijke handboek van maar liefst zo'n 270 pagina's. Hier zouden vele andere Logo-uitgevers een voorbeeld aan kunnen nemen.
Ook voor de Commodore Logo is een vloerschildpad te verkrijgen. Deze 'Valiant Turtle' is via de computer infrarood te besturen en kost als pakket (Turtle, interface, trafo en programmatuur) ongeveer 1600 gulden. Een en ander is te bestellen bij Malmberg/Fysica. Omdat de Valiant Turtle nogal prijzig is, heeft Malmberg ook een exemplaar (Lotje genaamd) dat aan scholen en dergelijke verhuurd wordt.
In de vergelijkingstabel vindt u onder Commodore een opsomming van de belangrijkste Commodore Terrapin Logo-primitieven.

LCN Logo

Het Logo Centrum Nijmegen heeft een zeer originele en hoogwaardige Logo ontwikkeld met LISP-achtige trekjes. De drempel voor deze Logo ligt, door de wat ingewikkelder extra's, iets hoger dan voor andere Logo-versies, maar het moet gezegd worden: het is een unieke, Nederlandstalige, Logo. Voor informatie hierover kunt u terecht bij het Logo Centrum Nijmegen (zie appendix 5). Deze Logo is niet opgenomen in de vergelijkingstabel.

MSX Logo

Deze LCSI Logo-versie is volledig Nederlandstalig, door Philips in modulevorm (ROM-cartridge) op de markt gebracht (begin 1985) en bij alle MSX (1 en 2) computers te gebruiken.
Met enige vertaling is alles wat in dit boek behandeld is direct in MSX Logo te gebruiken. De vergelijkingstabel kan hierbij voor u van grote waarde zijn.
De teken-, sproken- en woordenwereld zijn zeer volledig, de muziekwereld echter komt er tamelijk bekaaid af.

De Philips handleiding behandelt geen woord-lijstbewerkingen. Daarom heeft het Logo Centrum Ede er een Doe-het-zelf practicum voor gemaakt. Deze wordt, bij aankoop van de module in Ede, gratis bijgeleverd.

Spectrum Logo

Dit is de enige Logo-versie die op band wordt geleverd. Ondanks het feit dat u zeker vijf minuten moet wachten voor u aan de slag kunt, is de Spectrum Logo alleszins de moeite van het bekijken waard. Opvallend zijn de zeer mooie graphics (met een zeer hoge resolutie) en enkele handige printopdrachten als: PRINTON en COPYSCREEN. Ook zijn de opdrachten STARTROBOT en STOPROBOT beschikbaar voor het besturen van de vloerschildpad.
Spectrum Logo beschikt echter niet over een sprokenwereld en ook de muziekmogelijkheden zijn niet bepaald opzienbarend. Wat muziek betreft is standaard slechts de opdracht SOUND beschikbaar, waarmee (eenstemmig) de toonhoogte en de toonduur kunnen worden bepaald.
Ook deze Logo is een LCSI-produkt en de opdrachten zijn terug te vinden in de vergelijkingstabel.

We hebben de verschillende Logo-versies bijna allemaal gehad. Enkele zijn niet genoemd:

- De IBM Logo's (DR.Logo en LCSI Logo)
- DEC Logo (LCSI Logo)
- Kaypro en CP/M Logo-versies
- Schneider/Armstrad Logo.

Als u de vergelijkingstabel bekijkt, zult u ontdekken dat de verschillen meestal maar heel erg klein zijn. Het verschil zit soms in slechts een enkele letter: MEMBERP, MEMBERQ en MEMBER?. Met behulp van de vergelijkingstabel is het mogelijk om vrijwel alle procedures uit dit boek om te zetten naar uw eigen Logo-versie.

8.2 Vergelijkingstabel

In de vergelijkingstabel zijn lang niet alle grafische opdrachten voor de genoemde Logo-versies opgenomen. De meeste versies hebben veel meer grafische opdrachten en sprokenopdrachten. Zo heeft bijvoorbeeld de DR.Logo van Atari nog onder andere de volgende

GRAFISCHE OPDRACHTEN

Atari 8b	Atari 16b	Apple	BBC	Commodore	MSX	Spectrum
BK	BK	BK	BK	BK	at	BK
SETBG	SETBG	SETBG	SETBG	BG	kleurscherm	SETBG
CLEAN	CLEAN	CLEAN	CLEAN	CS	ms	CLEAN
-	-	-	-	DOUBLECOLOR	-	-
CS	CS	CS	CS	CS	ss	CS
BG	SF = SCREEN FACTS	BG	BG	BG	-	BG
PC/PN/PEN	TF = TURTLE FACTS	PC/PE	PC	-	pk	PC
FD	FD	FD	FD	FD	vt	FD
FS	-	FS	FS	FS	-	-
HEADING	HEADING	HEADING	HEADING	HEADING	richting	HEADING
HT	HT	HT	HT	HT	ws	HT
HOME	HOME	HOME	HOME	HOME	thuis	HOME
LEFT/LT	LEFT/LT	LEFT/LT	LEFT/LT	LEFT/LT	links/li	LEFT/LT
-	-	-	-	NODRAW	-	-
-	FENCE	FENCE	FENCE	NOWRAP	raam	FENCE
SETPC	SETPC	SETPC	SETPC	PC	zetpk	SETPC
PD	PD	PD	PD	PD	pn	PD
PU	PU	PU	PU	PU	po	PU
RIGHT/RT	RIGHT/RT	RIGHT/RT	RIGHT/RT	RIGHT/RT	rechts/re	RIGHT/RE
SETH	SETH	SETH	SETH	SETH	richt	SETH
SETSH	-	-	SETSH	SETSH	zetv	-
SETX	SETX	SETX	SETX	SETX	zetx	SETX
SETPOS	SETPOS	SETPOS	SETPOS	SETXY	zetpos	SETPOS
SETY	SETY	SETY	SETY	SETY	zety	SETY
SHAPE	-	-	-	SHAPE	-	-
ST	ST	ST	ST	ST	ts	ST
-	-	-	-	SINGLECOLOR	-	-
SS	-	SPLITSCREEN	SS	SPLITSCREEN	-	-
TELL	-	-	TELL	TELL	zeg	-
TS	-	TEXT-SCREEN	TS	TEXT-SCREEN	-	TS
-	TOWARDS	TOWARDS	TOWARDS	TOWARDS	naar	TOWARDS
WHO	-	-	WHO	WHO	wie	-

Atari 8b	Atari 16b	Apple	BBC	Commodore	MSX	Spectrum
WRAP	WRAP	WRAP	WRAP	WRAP	wikkel	WRAP
XCOR	XCOR	XCOR	XCOR	XCOR	xcoor	XCOR
YCOR	YCOR	YCOR	YCOR	YCOR	ycoor	YCOR
PE	PE	PE	PE	PENERASE	gum	PE
POS	POS	POS	POS	-	plaats	POSITION
-	-	DOT	DOT	-	punt	DOT

WOORD- EN LIJSTOPDRACHTEN

EQUALP	EQUALP	EQUALP	-	-	gelijkp	EQUALP
BF	BF	BF	BF	BF	me (mineerste)	BF
BL	BL	BL	BL	BL	ml (minlaatste)	BL
COUNT	COUNT	COUNT	COUNT	COUNT	tel	COUNT
EMPTYP	EMPTYP	EMPTYP	EMPTYQ	EMPTY?	leegp	EMPTYP
FIRST	FIRST	FIRST	FIRST	FIRST	eerste	FIRST
FPUT	FPUT	FPUT	FPUT	FPUT	ezet	FPUT
-	ITEM	ITEM	ITEM	ITEM	-	ITEM
LAST	LAST	LAST	LAST	LAST	laatste	LAST
LIST	LIST	LIST	LIST	LIST	lijst	LIST
LISTP	LISTP	LISTP	LISTQ	LIST?	lijstp	LISTP
LPUT	LPUT	LPUT	LPUT	LPUT	lzet	LPUT
MEMBERP	MEMBERP	MEMBERP	MEMBERQ	MEMBER?	-	MEMBERP
SE	SE	SE	SENTENCE	SE	zin	SE
WORD	WORD	WORD	WORD	WORD	woord	WORD
WORDP	WORDP	WORDP	WORDQ	WORD?	woordp	WORDP

PROCEDURES SCHRIJVEN EN HET GEBRUIK VAN VARIABELEN

Atari 8b	Atari 16b	Apple	BBC	Commodore	MSX	Spectrum
-	DEFINE	-	DEFINE	DEFINE	definieer	DEFINE
EDIT	EDIT	EDIT	EDIT	EDIT	edit	EDIT
END	END	END	END	END	eind	END
ERASE	ERASE	ERASE	ERASE	ERASE	gum	ERASE
-	TEXT	-	TEXT	TEXT	tekst	TEXT
TO	TO	TO	TO	TO	leer	TO
MAKE	MAKE	MAKE	MAKE	MAKE	maak	MAKE
THING	THING	THING	THING	THING	ding	THING
NAMEP	NAMEP	-	THINGQ	THING?	naamp	NAMEP
AND	AND	ALLOF	AND	ALLOF	en	AND
OR	OR	ANYOFF	OR	ANYOFF	of	OR
-	-	-	-	ELSE	-	-
IF	IF	IF	IF	IF	als	IF
-	IFF	IFF	IFFALSE	IFF	-	-
-	IFT	IFT	IFTRUE	IFT	-	-
NOT	NOT	NOT	NOT	NOT	niet	NOT
-	TEST	TEST	TEST	TEST	-	-
-	-	-	-	THEN	-	-
TRUE	TRUE	TRUE	TRUE	TRUE	waar	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	vals	FALSE
OP	OP	OP	OP	OP	gt	OP
REPEAT	REPEAT	REPEAT	REPEAT	REPEAT	herhaal	REPEAT
RUN	RUN	-	RUN	RUN	vooruit	RUN
STOP	STOP	STOP	STOP	STOP	stop	STOP

REKENKUNDIGE BEWERKINGEN

Atari 8b	Atari 16b	Apple	BBC	Commodore	MSX	Spectrum
-	ARCTAN	ARCTAN	ATN	ATAN	arctan	ARCTAN
COS	COS	COS	COS	COS	cos	COSINE
INT	INT	INT	INT	INTEGER	rondaf	INT
NUMBERP	NUMBERP	NUMBERP	NUMBERQ	NUMBER?	getalp	NUMBERP
-	QUOTIENT	QUOTIENT	QUOTIENT	QUOTIENT	quotient	-
RANDOM	RANDOM	RANDOM	RANDOM	RANDOM	gok	RANDOM
RERANDOM	RERANDOM	-	RERANDOM	-	-	-
REMAINDER	REMAINDER	REMAINDER	REMAINDER	REMAINDER	-	REMAINDER
ROUND	ROUND	ROUND	ROUND	ROUND	rondaf	ROUND
SIN	SIN	SIN	SIN	SIN	sinus	SINE
SQRT	SQRT	SQRT	SQRT	SQRT	wortel	SQRT
PRODUCT	PRODUCT	PRODUCT	PRODUCT	PRODUCT	product	PRODUCT
SUM	SUM	SUM	SUM	SUM	-	SUM
TAN	TAN	TAN	TAN	TAN	-	-
+	+	+	+	+	+	+
–	–	–	–	–	–	–
*	*	*	*	*	*	*
/	/	/	/	/	/	/
<	<	<	<	<	<	<
>	>	>	>	>	>	>
=	=	=	=	=	=	=

TENSLOTTE

ASCII	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII
CHAR	CHAR	CHAR	CHAR	CHAR	kar	CHAR
CT	CT	CLEARTEXT	CT	CLEARTEXT	tw	CLEARTEXT
SETCURSOR	-	SETCURSOR	SETCURSOR	CURSOR	zetcursor	SETCUR
SHOW	SHOW	SHOW	SHOW	FPRINT	laatzien	SHOW
JOY	-	-	JOY	JOYSTICK	joy	-
JOYB	-	-	JOYB	JOYBUTTON	-	-
PADDLE	-	-	PADDLE	PADDLE	-	-
PADDLEB	-	-	PADDLEB	-	-	-

opdrachten: BOX, CIRCLE, ELLIPSE, FILL, SETZOOM.

Een moeilijkheid is soms ook dat opdrachten met min of meer dezelfde naam bij verschillende Logo-versies een andere werking hebben. Dit is bijvoorbeeld het geval met SHAPE (Atari/Commodore) en SH(BBC).

Waar opdrachten ongelijke werking hebben of niet aanwezig zijn staat een streepje: - .

Ook de opdrachten die we bij het manipuleren met woorden en lijsten gebruiken zijn lang niet allemaal opgenomen. Iedere Logo-versie voegt hier zijn eigen specifieke opdrachten aan toe. Het is aardig om eens naar de DR.Logo van Atari (16 bits) te kijken. Deze Logo-versie beschikt over zeer sterke opdrachten voor de lijstbewerking. Om eens enkele te noemen:

SHUFFLE is een DR.Logo-opdracht die alle elementen van de gegeven lijst op RANDOM-wijze, dus willekeurig, rangschikt;

WHERE is een opdracht die de plaats van een element in een lijst aangeeft.

Maar zoals u weet, met enige kennis van zaken is bijna iedere opdracht voor iedere Logo-versie zelf te schrijven.

Wellicht heeft u zich afgevraagd waarom er geen sproken- en muziekopdrachten in de vergelijkingstabel zijn genoemd. Deze beide horen toch ook bij Logo?

Dat is waar, maar we moeten constateren dat deze mogelijkheden lang niet in iedere Logo-versie te vinden zijn. Spectrum Logo bijvoorbeeld kent geen sprokenwereld.

Logo-versies 'met' muziek en sproken (Atari 8 bits, Apple, BBC, Commodore en MSX) zijn in werking en opzet zo verschillend dat het vrijwel onmogelijk is deze te vergelijken. Om maar eens iets te noemen: de Commodore Logo heeft geweldige muziekmogelijkheden in tegenstelling tot bijvoorbeeld de Atari 8-bits en MSX Logo. DR. Logo van Atari 16-bits heeft helemaal geen muziekmogelijkheden. Wat sproken betreft hebben zowel Commodore als MSX prima mogelijkheden, maar deze liggen in werking en opzet zover uit elkaar, dat ze niet te vergelijken zijn.

Zo heeft iedere Logo zijn bijzonderheden. Geen enkele Logo is slecht te noemen. Iedereen kan experimenteren met de mogelijkheden van zijn eigen Logo. Al werkende met Logo maakt u uw eigen opdrachten en schrijft u zo uw eigen taal 'op maat'. Dat is het bijzondere van een taal als Logo. Maar dat had u natuurlijk al begrepen.

APPENDICES

A.1 Lijst van de meestgebruikte Logo-primitieven

AND	- PR AND "TRUE "TRUE Geeft TRUE als waarde als beide invoergegevens TRUE zijn
ASCII	- PR ASCII "C Geeft de ASCII-code voor het genoemde teken
ASK	- ASK 1 [FD 40] Laat de genoemde Turtle(s) de gegeven opdrachten uitvoeren. Beïnvloedt niet de met TELL aangeroepen Turtles
BACK / BK	- BK 100 Laat de laatst aangeroepen Turtle(s) het gegeven aantal stappen achteruit gaan
BG	- PR BG Geeft het nummer van de huidige achtergrondkleur
BUTFIRST / BF	- PR BF [1 2 3] Geeft als resultaat de elementen van de gegeven lijst op het eerste na
BUTLAST / BL	- PR BL [1 2 3] Geeft als resultaat de elementen van de gegeven lijst op het laatste na
.CALL	- .CALL 1536 Roept een machinetaalroutine aan beginnende op het gegeven adres
CATALOG	- CATALOG "D: Geeft de inhoudsopgave van de diskette.
CHAR	- PR CHAR 66 Geeft het teken dat bij de genoemde ASCII-code hoort
CLEAN	- CLEAN Geeft een schoon grafisch scherm, maar beïnvloedt niet de snelheid, richting en positie van de Turtle(s)

COLOR	- PR COLOR Geeft als resultaat de kleur van de laatst aangeroepen Turtle
COND	- IF COND 3 [PR [Hallo]] Geeft TRUE als waarde als aan de conditie voldaan wordt. Zie hiervoor de Collision Tabel
COS	- PR COS 90 Geeft als resultaat de cosinus van het genoemde aantal graden
COUNT	- PR COUNT [a d e w] Geeft als resultaat het aantal elementen in de gegeven lijst of woord
CS	- CS Maakt het grafische scherm schoon. Zet de Turtle(s) in 'thuis'positie, dat is 0, 0 richting 0
CT	- CT Maakt het tekstscherm schoon
.DEPOSIT	- .DEPOSIT 53256 3 Geeft het eerstgenoemde geheugenadres de laatstgenoemde waarde
EACH	- EACH [SETH 90] Laat iedere Turtle afzonderlijk de genoemde opdrachten uitvoeren
EDIT	- EDIT "PROCEDURE Start de Logo Editor met de genoemde procedure
EDNS	- EDNS Start de Logo Editor met alle in het werkgeheugen aanwezige variabelen
EDSH	- EDSH 1 Start de Shape Editor met het genoemde 'vorm'-nummer
EMPTYP	- IF EMPTYP :V [PR [Leeg !!!]] Kijkt of de/het genoemde lijst/woord wellicht 'leeg' is. Geeft dan als waarde TRUE
END	- END Besluit een procedure

EQUALP	- IF EQUALP :a :b [PR [Gelijk !!!!]] Kijkt of beide invoergegevens (dit kan een lijst/ woord/getal zijn) wellicht 'gelijk' zijn. Geeft dan als waarde TRUE
ERALL	- ERALL Maakt het gehele werkgeheugen leeg
ERASE	- ERASE "PROCEDURE Haalt de genoemde procedure(s) uit het werkgeheugen en 'wist' ze. Meerdere procedures moeten in een lijst geplaatst worden. Bijvoorbeeld ERASE [een twee drie vier]
ERF	- ERF "D:LOGOFILE.EXT Erase File; deze opdracht 'wist' het genoemde bestand van de diskette
ERN	- ERN "antw 'Wist' de genoemde variabele(n). Moeten er meerdere variabelen worden gewist, plaats ze dan in een lijst
ERNS	- ERNS 'Wist' alle variabelen uit het werkgeheugen
ERPS	- ERPS 'Wist' alle procedures uit het werkgeheugen
.EXAMINE	- PR .EXAMINE 53257 Geeft als resultaat de waarde van het genoemde geheugenadres
FIRST	- PR FIRST [a b c] Geeft als resultaat het eerste element van de gegeven invoer. Dit kan een woord zijn, een getal of een lijst
FORWARD / FD	- FD 100 Laat de laatst aangeroepen Turtle(s) het gegeven aantal stappen vooruitgaan
FPUT	- PR FPUT "een [twee] Geeft als resultaat een lijst waarbij het genoemde element voorin de genoemde lijst wordt geplaatst
FS	- FS Maakt het gehele scherm (Full Screen) geschikt voor grafische weergave

GETSH	- MAKE "vorm GETSH 1 Geeft als resultaat een lijst van zestien getallen; dit zijn de gegevens van de genoemde vorm
HEADING	- PR HEADING Geeft als resultaat de richting van de laatst aangeroepen Turtle
HOME	- HOME Plaatst de aangeroepen Turtle(s) in uitgangspositie (0 0) in de richting 0
HT	- HT Maakt de aangeroepen Turtle(s) onzichtbaar (Hide Turtle)
IF	- IF :z < 20 [VIERKANT] [STOP] Laat, indien aan de genoemde voorwaarde wordt voldaan, de opdracht(en) uit de eerste lijst uitvoeren. Wordt aan de genoemde voorwaarde niet voldaan, dan wordt/worden de opdracht(en) uit de tweede lijst uitgevoerd
INT	- PR INT 53.275 Geeft als resultaat de integere waarde van het genoemde getal
JOY	- MAKE "R JOY 0 Geeft als resultaat de positie van de genoemde joystick
JOYB	- IF JOYB 0 [PR [Ingedrukt !!]] Geeft als waarde TRUE indien de rode knop van genoemde joystick wordt ingedrukt
KEYP	- IF KEYP [PR [Een toets ingedrukt !!!]] Geeft als waarde TRUE indien een toets van het toetsenbord wordt ingedrukt
LAST	- PR LAST "huis Geeft als resultaat het laatste element van de genoemde lijst of het genoemde woord
LEFT / LT	- LT 90 Draait de aangeroepen Turtle(s) het gegeven aantal graden linksom
LIST	- PR LIST "een "twee Geeft als resultaat een lijst, bestaande uit de genoemde invoergegevens. Deze invoergegevens mogen zelf ook lijsten zijn

LISTP	- IF LISTP :L [PR [Dit is een lijst]] Geeft als waarde TRUE als het genoemde invoergegeven een lijst is
LOAD	- LOAD "C: Laadt een bestand van cassette ("C:) of diskette ("D:). Bij het laden van diskette moet een bestandsnaam worden genoemd
LPUT	- PR LPUT "twee [een] Geeft als resultaat een lijst waarbij het genoemde element achter in de genoemde lijst wordt geplaatst
MAKE	- MAKE "x :x + 1 Maakt een variabele en geeft deze de genoemde waarde
MEMBERP	- PR MEMBERP "a [b c d a] Geeft als waarde TRUE indien de genoemde invoer deel uitmaakt van de genoemde lijst
NAMEP	- PR NAMEP :a Geeft als waarde TRUE indien genoemde variabele een waarde heeft. Dat wil zeggen als genoemde variabele bestaat!
NODES	- PR NODES Geeft als resultaat de nog beschikbare geheugenruimte
NOT	- PR NOT NUMBERP "HALLO Geeft als waarde TRUE indien aan de voorwaarde NIET wordt voldaan. Geeft als waarde dus FALSE als aan de voorwaarde wel wordt voldaan. Het voorbeeld geeft dus TRUE, want een woord is GEEN getal
NUMBERP	- PR NUMBERP 777 Geeft als waarde TRUE indien genoemde invoer een getal is
OR	- PR OR "TRUE "FALSE Geeft als waarde TRUE indien aan een van beide voorwaarden wordt voldaan
OUTPUT / OP	- OP SUM 776 883 Kan alleen 'in' een procedure worden gebruikt en 'onthoudt' als het ware het gevolg van de gegeven opdracht

PC	- PR PC 1 Geeft als resultaat de kleur van de genoemde pen
PE	- PE Maakt van de pen van de aangeroepen Turtle(s) een vlakgum
PEN	- PR PEN Geeft als resultaat de toestand van de pen van de aangeroepen Turtle(s). Dit kan zijn PD, PU, PE of PX
PENDOWN / PD	- PD Laat de tekenpen van de aangeroepen Turtle(s) zakken
PENUP / PU	- PU Haalt de tekenpen van de aangeroepen Turtle(s) omhoog
PN	- PR PN Geeft als resultaat het nummer van de gebruikte pen bij de laatst aangeroepen Turtle(s)
PO	- PO "PROCEDURE Beeldt op het beeldscherm de definitie van de genoemde procedure af. Wanneer meerdere procedures worden gevraagd [ ] gebruiken
POALL	- POALL Beeldt op het beeldscherm zowel de procedures als de variabelen af
POD	- POD 0 Geeft de When-Demon aan bij genoemde Collision. Met andere woorden: 'wat gebeurt er indien, in dit geval, Collision 0 plaatsvindt?'
PODS	- PODS Geeft alle actieve When-Demons aan
PONS	- PONS Beeldt op het beeldscherm alle variabelen af plus de waarde die zij hebben
POPS	- POPS Beeldt op het beeldscherm alle procedures af plus hun definities

POS	- PR POS Geeft als resultaat de coördinaten van de laatst aangeroepen Turtle(s)
POTS	- POTS Beeldt op het beeldscherm alle namen van de in het werkgeheugen staande procedures af (alleen hun namen dus!)
.PRIMITIVES	- .PRIMITIVES Beeldt op het beeldscherm alle Logo-primitieven af
PRINT / PR	- PR [Hallo Allemaal] Beeldt op het beeldscherm de gegeven invoer af. De PRINT-opdracht kan ook gevolgd worden door "
PRODUCT	- PR PRODUCT 5 6 Geeft als resultaat het produkt van beide getallen
PUTSH	- PUTSH 1 :huis Geeft aan de genoemde vorm de in een variabele vastgelegde gegevens
PX	- PX Maakt van de pen van de laatst aangeroepen Turtle(s) zowel een tekenpen (op de lege vlakken) als een uitwispen (op reeds getekende vlakken)
RANDOM	- PR RANDOM 10 Geeft aan de genoemde vorm de (in een variabele vastgelegde) gegevens
RC	- MAKE "keus RC Geeft als resultaat het laatst ingedrukte teken
RECYCLE	- RECYCLE Maakt het werkgeheugen schoon door alle typefouten enzovoort te wissen
REMAINDER	- PR REMAINDER 10 3 Geeft als resultaat het 'restgetal' na het eerste getal door het tweede te hebben gedeeld
REPEAT	- REPEAT 7 [FD 8 RT 45] Herhaalt de gegeven opdracht het genoemde aantal malen
RERANDOM	- RERANDOM REPEAT 2 [PR RANDOM 5] 'Onthoudt' de RANDOM-getallen en hun volgorde. Later kunnen deze dan opnieuw worden gebruikt

RIGHT / RT	- RT 90 Draait de aangeroepen Turtle(s) het aangegeven aantal graden rechtsom
RL	- MAKE "naam RL Geeft als resultaat (en plaatst in een lijst) het woord, de woorden of de zin die volgt
ROUND	- PR ROUND 5.0045 Geeft als resultaat het naar het dichtstbijzijnde gehele getal afgeronde invoergetal
RUN	- RUN [VIERKANT] Voert in een procedure de gegeven opdrachten of de gegeven procedure uit alsof het in de 'Direct Mode' werd ingevoerd
SAVE	- SAVE "P: Schrijft de inhoud van het werkgeheugen weg naar "C: cassette "P: printer "D:FILENAAM diskette Zowel procedures als variabelen worden weggeschreven
SE	- PR SE "hallo "daar Geeft als resultaat een zin, die de gegeven invoer bevat
SETBG	- SETBG 77 Geeft de achtergrond een kleur als aangegeven
SETC	- SETC 76 Geeft de laatst aangeroepen Turtle(s) de aangegeven kleur
SETCURSOR	- SETCURSOR [10 2] Zet de cursor op de aangegeven plaats
SETENV	- SETENV 0 126 Zet de Envelope Shaper voor het genoemde (eerste invoergegeven) geluidskanaal op de waarde van het tweede invoergegeven
SETH	- SETH 270 Draait de aangeroepen Turtle(s) in de aangegeven richting
SETPC	- SETPC 0 120 Verandert de genoemde tekenpen (0, 1 of 2) in de aangegeven kleur

SETPN	- SETPN 1 Geeft de laatst aangeroepen Turtle(s) de genoemde pen
SETPOS	- SETPOS [0 40] Plaatst de laatst aangeroepen Turtle(s) op de genoemde coördinaten
SETREAD	- SETREAD "D:FILENAAM Geeft als resultaat het genoemde bestand uit het genoemde randapparaat. Het bestand kan pas gelezen worden met de opdracht RC of RL
SETREAD [ ]	- SETREAD [ ] Sluit het met SETREAD "D:FILENAAM geopende bestand
.SETSCR	- .SETSCR 1 Bepaalt de resolutie van het beeldscherm
SETSH	- SETSH 2 Geeft de laatst aangeroepen Turtle(s) de genoemde vorm
SETSP	- SETSP 150 Geeft de laatst aangeroepen Turtle(s) de genoemde snelheid. Opmerking. De invoer mag ook negatief zijn. Hierdoor beweegt/bewegen de Turtle(s) zich achteruit
SETWRITE	- SETWRITE "P Opent een bestand naar het genoemde randapparaat en stuurt daarheen alle op het beeldscherm verschijnende informatie. Met SETWRITE "P: en CATALOG "D: wordt dus de inhoudsopgave van de diskette afgedrukt
SETWRITE [ ]	- SETWRITE [ ] Sluit het met SETWRITE "D:FILENAAM (of SETWRITE "P:) geopende bestand
SETX	- SETX 40 Zet de laatst aangeroepen Turtle(s) op de gegeven X-coördinaat
SETY	- SETY -100 Zet de laatst aangeroepen Turtle(s) op de gegeven Y-coördinaat

SHAPE	- PR SHAPE Geeft als resultaat het 'Shape' (vorm-)nummer van de laatst aangeroepen Turtle(s)
SHOW	- SHOW [A C E] Beeldt de gegeven invoer op het beeldscherm af. Indien de invoer een lijst is, wordt deze met haakjes afgedrukt
SHOWNP	- IF SHOWNP [HT] Geeft als waarde TRUE indien de laatst aangeroepen Turtle(s) zichtbaar is/zijn
SIN	- PR SIN 90 Geeft als resultaat de sinus van de gegeven hoek
SPEED	- PR SPEED Geeft als resultaat de snelheid van de laatst aangeroepen Turtle(s)
SQRT	- PR SQRT 16 Geeft als resultaat de (vierkants)wortel van het genoemde getal
SS	- SS Geeft een gedeeld beeldscherm (tekst en graphics)
ST	- ST Maakt de aangeroepen Turtle(s) zichtbaar
STOP	- IF :zijde > 20 [STOP] Stopt de procedure en kan dus alleen IN een procedure worden gebruikt
SUM	- PR SUM 999 111 Geeft als resultaat de som van beide genoemde getallen
TELL	- TELL [0 1 2 3] ST Roept genoemde Turtle(s) op de erna volgende opdrachten uit te voeren
THING	- PR THING :v Geeft als resultaat de waarde van de genoemde variabele
TOOT	- TOOT 0 261 10 30 Opent het geluidskanaal (eerste getal), produceert daar een toon van de genoemde frequentie (tweede getal) op het genoemde volumeniveau (derde getal) gedurende de genoemde tijd (vierde getal)

TS	- TS Maakt het gehele beeldscherm beschikbaar voor tekst
TYPE	- TYPE "a TYPE "b Drukt de gegeven invoer op het beeldscherm af maar springt daarna niet naar een nieuwe regel. De cursor blijft dus aan het einde van de afgedrukte regel staan
WAIT	- WAIT 60 Geeft een pauze van het genoemde aantal tijdseenheden. De gebruikte eenheid is hier 1/60 seconde
WHEN	- WHEN 3 [HT] Maakt genoemde When-Demon actief en laat hem de in de lijst genoemde opdrachten uitvoeren - WHEN 3 [ ] Maakt de genoemde When-Demon inactief
WHO	- PR WHO Geeft als resultaat de actieve/laatst aangeroepen Turtle(s)
WINDOW	- WINDOW Maakt van het grafische scherm een 'WINDOW' (venster). Laat dus maar een klein gedeelte zien van het Turtle 'tekenterrein'
WORD	- PR WORD "aard "bij Geeft als resultaat een woord bestaande uit de genoemde invoergegevens
WORDP	- PR WORDP "woord Geeft als waarde TRUE indien de invoer een woord is
WRAP	- WRAP Maakt het Turtle 'tekenterrein' zo groot als het beeldscherm. Laat de Turtle, indien deze van het scherm verdwijnt, er aan de andere kant weer op komen
XCOR	- PR XCOR Geeft als resultaat de X-coördinaat van de laatst aangeroepen Turtle(s)
YCOR	- PR YCOR Geeft als resultaat de Y-coördinaat van de laatst aangeroepen Turtle(s)

A.2 De Logo-foutmeldingen

Opmerking. Ook onder de foutmeldingen vertonen de diverse Logo-versies minieme verschillen. Kijk dit goed na voor uw eigen Logo.

- OUT OF SPACE

  De beschikbare geheugenruimte is vol. Geef de opdracht RECYCLE (of de vergelijkbare opdracht voor uw Logo-versie) of haal enkele procedures of variabelen uit het geheugen.

- I DON'T KNOW HOW TO ...

  Een niet-gedefinieerde procedure is aangeroepen.

- ... HAS NO VALUE

  U heeft een variabele gebruikt die geen waarde heeft. Geef deze variabele een waarde met bijvoorbeeld de opdracht MAKE.

- NOT ENOUGH INPUTS TO ...

  U heeft een procedure aangeroepen of een opdracht gebruikt zonder de hiervoor benodigde invoergegevens.

- YOU DON'T SAY WHAT TO DO WITH ...

  Een Logo-object (woord, lijst of getal) is gebruikt zonder een opdracht.

- TO MUCH INSIDE () 'S

  In een Logo-opdracht zijn teveel haakjes gebruikt. Zoek een andere oplossing voor deze opdracht. Maak er bijvoorbeeld twee opdrachten van.

- ... DIDN'T OUTPUT TO ...

  U bent de invoer vergeten in de ene procedure (of opdracht) en daardoor werkt ook de tweede procedure (of opdracht) niet.

- ... IS A PRIMITIVE

  U hebt een ongeoorloofde procedurenaam gekozen. Deze is namelijk een Logo-opdracht en kan dus niet als procedurenaam worden gebruikt.

- ... DOESN'T LIKE ... AS INPUT

  U hebt voor een opdracht een ongeoorloofde invoer gebruikt. Bijvoorbeeld PR SUM HUIS 990.

- UNEXPECTED ')'

  U hebt een sluithaakje) gebruikt zonder dat er een openingshaakje (was.

- NUMBER TOO BIG

  U laat een rekenkundige bewerking uitvoeren waarvan het resultaat groter is dan 1E98 = 10^{98} of kleiner is dan 1E-98 = 10^{-98}.

- ... IS ALREADY DEFINED

  U heeft al een procedure geschreven met de zojuist opgegeven naam. Kies dus een andere naam of haal de andere procedure uit het geheugen.

- ... IS NOT TRUE OR FALSE

  U heeft bij AND, OR, IF of NOT een invoergegeven gebruikt dat onmogelijk TRUE of FALSE kan zijn.

- I CAN'T OPEN ...

  U heeft de opdracht LOAD, SAVE, SETREAD of SETWRITE onjuist gebruikt. De oorzaak van deze foutmelding kan bijvoorbeeld zijn:
  - u heeft Logo gestart op een DOS-besturingssysteem met enkele dichtheid en probeert nu een bestand te lezen van een schijf met dubbele dichtheid;
  - u wilt een bestand laden zonder aan te geven waarvan. Bijvoorbeeld LOAD "PROGRAM.

  In de meeste gevallen geeft Logo precies aan waar de fout zit.

- FILE ... NOT FOUND

  U wilt een bestand laden (met LOAD of SETREAD) dat niet te vinden is.

- YOU'RE AT TOPLEVEL

  U heeft de opdracht STOP of OUTPUT gebruikt in de Direct Mode. Buiten een procedure om kunnen deze opdrachten niet worden gebruikt.

- STOPPED !

  U heeft de BREAK-toets gebruikt. Hiermee onderbrak u de lopende procedure of de gebruikte opdracht.

Opmerking. Een nieuwe taal, zoals Logo, leren is een zaak van vallen en opstaan, van fouten maken en fouten herstellen. In het Engels is daar een mooi woord voor: DEBUGGEN - de BUGS, de fouten eruit halen. Logo zou niet een van de meest gebruikersvriendelijke talen zijn als ook dit niet soepel ging.

Het systeem van foutmeldingen bij Logo is ideaal. Het zijn geen foutmeldingen met cijfers maar met duidelijke zinnen die u vertellen wat er aan de hand is. Toegegeven, het is even wennen, maar na een tijdje werkt het vlot.
Logo geeft tijdens het uitvoeren van een procedure of een opdracht precies aan wat u fout gedaan heeft (doet).
Omdat Logo - in tegenstelling tot bijvoorbeeld BASIC - niet tijdens het programmeren, maar tijdens het draaien de fouten aangeeft, is het essentieel om bij het schrijven van een groot programma een en ander op te splitsen in (vele) subprocedures.
In deze subprocedures kunt u dan afzonderlijk de fouten gaan opzoeken. En ze herstellen natuurlijk!

A.3 Logo en machinetaal

Heel bewust is in dit boek weinig geschreven over het 'direct werken met geheugenadressen' in Logo. Dit is namelijk nogal merkgebonden. Dat wil zeggen dat de geheugenadressen per computermerk verschillend zijn. Toch is het goed in deze appendix even kort in te gaan op de machinetaal voor Atari LCSI Logo. Ook als u over een andere Logo-versie beschikt (en de genoemde geheugenadressen dus niet kunt gebruiken) kunt u zich wellicht een beeld vormen van wat er ook in machinetaal bij Logo mogelijk is.

Atari LCSI Logo beschikt over een PEEK- en een POKE-opdracht, respectievelijk .EXAMINE en .DEPOSIT geheten. Met .EXAMINE 'kijken' we in een geheugenadres om er een waarde af te lezen. Met .DEPOSIT 'zetten' we een waarde in een geheugenadres. Daarom vraagt .EXAMINE één invoergegeven (de geheugenplaats) en .DEPOSIT twee (de geheugenplaats en de 'aldaar te plaatsen' waarde).

Atari-gebruikers moeten hierbij goed om de . voor de opdracht denken. De stip is ervoor om aan te geven dat het gebruik van deze opdrachten niet zonder gevaar is. Door een bepaalde waarde in een geheugenadres te stoppen, kan de computer vastlopen en kan het programma verloren gaan.

De 8-bits processor, het hart van de Atari-computer, kan 65536 geheugenplaatsen aansturen. Elk van deze geheugenplaatsen kan acht bits 'informatie' in zich opnemen. Voor ieder bit zijn er maar twee mogelijkheden: 'aan' of 'uit'. Als een bit 'aan' is, spreken we van spanning en schrijven een 1. Is het bit 'uit' dan is er geen spanning en schrijven we een 0. Deze acht bits (genummerd 0-7) heten samen een byte.

Wanneer we nu de waarde van ieder bit weten (dus of het aan/uit is) en we noteren hiervoor een 1 of een 0, dan krijgen we een binair getal van acht cijfers, bijvoorbeeld 10001111. Iedere byte nu kan een waarde hebben van 0 t/m 255 (decimaal) ofwel 00 t/m FF (hexadecimaal). En dat is logisch: zijn alle bits 'aan', dan is de 'bit-waarde' 11111111 (en dat is het decimale getal 255); zijn alle bits 'uit', dan is de 'bit-waarde' 00000000 (en dat is het decimale getal 0).

Het is niet zinvol hier alle 65536 geheugenplaatsen op te noemen, maar hier volgen enkele belangrijke Atari-geheugenplaatsen:

18, 19 en 20	de interne klok van de Atari. Straks krijgt u een kort voorbeeldprogramma over deze geheugenplaatsen
77	Atari-gebruikers weten dat, wanneer zij hun toetsenbord enige tijd niet hebben aangeraakt, hun beeldscherm steeds van kleur verandert. Iedere keer dat u het toetsenbord gebruikt wordt namelijk op deze

plaats een 0 gezet. Wordt het toetsenbord niet gebruikt, dan wordt de waarde iedere vier seconden met 1 opgehoogd. Bereikt de inhoud van adres 77 nu de waarde 128, dan gaat de computer over in de zogenaamde 'Atract'.mode en verandert de kleur van het beeldscherm dus steeds. Om dit nu bij (recursieve) programma's met een joystick te voorkomen, moet het programma minimaal één keer per acht minuten de waarde 0 in adres 77 stoppen. Het is dus niet voldoende (vooraf) eenmaal .DEPOSIT 77 0 te geven. Beter is het deze opdracht 'recursief' in te bouwen.

82, 83 Kolomwaarden voor respectievelijk de linker- en de rechterrand van het beeldscherm. Gewoonlijk vinden we hier de waarden 2 en 39.

559 .DEPOSIT 559 0 schakelt de beeldschermprocessor uit. U kunt dit zien door het 'zwart' worden van het beeld. Het beeldschermgeheugen wordt echter niet leeggemaakt. De opdracht kan nuttig zijn bijvoorbeeld bij het maken van lange berekeningen. Vergeet niet om naderhand in dit adres weer de 'oude' waarde terug te zetten.

704 Kleurregister voor Turtle 0.
705 Kleurwaarde voor Turtle 1.
706 Kleurwaarde voor Turtle 2.
707 Kleurwaarde voor Turtle 3.

731 Bepaalt het geluid bij het indrukken van een toets. Iedere waarde ongelijk aan nul verwijdert de 'keyboard-click'. Bijvoorbeeld .DEPOSIT 731 1.

755 Dit is het register voor de weergave van de tekens.
Bij 0 worden alle inverse tekens als normaal behandeld. Daardoor wordt de cursor 'onzichtbaar'.
Bij 1 worden alle inverse tekens als spaties weergegeven.
2 is de standaardwaarde van dit adres.
Bij 3 worden alle inverse tekens als inverse spaties weergegeven.

767 Start- en stopaanduiding voor de beeldschermweergave. De normale waarde van dit adres is 0. Door het indrukken van de toetsen CTRL en 1 komt hier de waarde 255 te staan. We noemen dit een invertering (omkering) (van 00000000 tot 11111111).

53256	Bepaalt de 'breedte' van Turtle 0. De standaardwaarde is 0 of 2. Zetten we hier de waarde 1 of 3, dan maken we de Turtle 'breder'.
53257	Idem voor Turtle 1.
53258	Idem voor Turtle 2.
53259	Idem voor Turtle 3.
53770	Uit dit geheugenadres kunnen toevalswaarden (tussen 0 en 255) worden gelezen. Dit heeft dus hetzelfde resultaat als een RANDOM-opdracht.

Zoals gezegd, dit zijn slechts enkele van de 65536 geheugenadressen. Voor degenen onder u die zich willen verdiepen in deze materie is er voldoende literatuur. Voor de Atari kan genoemd worden:
- *Mapping the Atari*, door Ian Chadwick, uitgegeven door Compute Books, ISBN 0 87455 004 1.
- *Atari Peeks en Pokes*, Koch, uitgegeven door Data Becker/Bruna.

Tot slot volgt hier nog het programma: de Logo-klok. De klok, gebaseerd op de geheugenplaatsen 18, 19 en 20, is een bekend Atari-programma. Logo-klok maakt van het beeldscherm een digitale klok met dagen, uren, minuten en seconden. Het programma heeft reeds in vele boeken en tijdschriften gestaan maar nog nooit in een Logo-uitvoering.

Het programma 'lklok' maakt slechts gebruik van één hoofd- en twee subprocedures. Het is ook een aardig programma om mee te experimenteren.

U vindt de listing op de volgende bladzijde.

```
TO lklok
CT
MAKE "n 0
nul 0
klok
END


TO klok
.DEPOSIT 752 1 TS
MAKE "t INT ( .EXAMINE 18 ) * 655536 + (
 .EXAMINE 19 ) * 256 + ( .EXAMINE 20 )
MAKE "T INT ( :t / 50 )
MAKE "D INT ( :T / 86400 )
MAKE "HH INT ( :T / 3600 )
MAKE "H :HH - :D * 24
MAKE "MM INT ( :T / 60 )
MAKE "M :MM - :D * 1440 - :H * 60
MAKE "S :T - :D * 86400 - :H * 3600 - :M
 * 60
SETCURSOR [7 2] PR [* * L o g o k l o k
* *]
SETCURSOR [5 4] PR ( SE :D "dag :H "uur
:M "min. :S "sec.
klok
END


TO nul :n
IF :n = "3 [STOP]
.DEPOSIT ( 18 + :n ) 0
nul :n + 1
END
```

A.4 ASCII-codetabel

0	♥										
1		21		41	)	61	=	81	Q	101	e
2		22		42	*	62	>	82	R	102	f
3		23		43	+	63	?	83	S	103	g
4		24		44	,	64	@	84	T	104	h
5		25		45	-	65	A	85	U	105	i
6		26		46	.	66	B	86	V	106	j
7		27		47	/	67	C	87	W	107	k
8		28	↑	48	0	68	D	88	X	108	l
9		29	↓	49	1	69	E	89	Y	109	m
10		30	←	50	2	70	F	90	Z	110	n
11		31	→	51	3	71	G	91	[	111	o
12		32		52	4	72	H	92	\	112	p
13		33	!	53	5	73	I	93	]	113	q
14		34	"	54	6	74	J	94	^	114	r
15		35	#	55	7	75	K	95	–	115	s
16	♣	36	$	56	8	76	L	96	♦	116	t
17		37	%	57	9	77	M	97	a	117	u
18		38	&	58	:	78	N	98	b	118	v
19		39	'	59	;	79	O	99	c	119	w
20		40	(	60	<	80	P	100	d	120	x

121	y	146		171	+	196	D	221	]	246	v
122	z	147		172	,	197	E	222	^	247	w
123		148		173	-	198	F	223	_	248	x
124		149		174	.	199	G	224		249	y
125		150		175	/	200	H	225	a	250	z
126		151		176	0	201	I	226	b	251	
127		152		177	1	202	J	227	c	252	
128		153		178	2	203	K	228	d	253	
129		154		179	3	204	L	229	e	254	
130		155	EOL	180	4	205	M	230	f	255	
131		156		181	5	206	N	231	g		
132		157		182	6	207	O	232	h		
133		158		183	7	208	P	233	i		
134		159		184	8	209	Q	234	j		
135		160		185	9	210	R	235	k		
136		161	!	186	:	211	S	236	l		
137		162	"	187	;	212	T	237	m		
138		163	#	188	<	213	U	238	n		
139		164	$	189	=	214	V	239	o		
140		165	%	190	>	215	W	240	p		
141		166	&	191	?	216	X	241	q		
142		167	'	192	@	217	Y	242	r		
143		168	(	193	A	218	Z	243	s		
144		169	)	194	B	219	[	244	t		
145		170	*	195	C	220	\	245	u		

A.5 Namen, adressen en tips

Wanneer u met vragen zit over het gebruik van Logo kunt u contact opnemen met:

Logo Centrum Nijmegen
Postbus 1408
6501 BK Nijmegen
Tel. 080 - 238130

of Logo Centrum Ede
Annalaan 96
6715 JC Ede
Tel. 08380 - 21306

De stuwende kracht achter het Logo Centrum Ede is Willem Heyster. Reeds eerder zijn de Logo-werkboekjes genoemd, die u hier kunt bestellen. Nieuw is het boek *Logologie, lekker puzzelen met de computer*. Dit boek is geschreven door Willem Heyster en uitgegeven door het Logo Centrum Ede. Logo is bij uitstek geschikt om het analytisch denken te bevorderen. Daarom daagt dit boek u uit allerlei constructies om te zetten in Logo-procedures.

Een aantal van die problemen trof u in dit boek reeds aan als illustratie.

Een stichting die Logo-kampen organiseert is:

Stichting Logo K
Postbus 1407
8011 PE Zwolle

Scholen die met vragen zitten kunnen natuurlijk ook contact opnemen met de verschillende schoolbegeleidingsdiensten of met de diverse regionale steunpunten onderwijs en informatietechnologie (R.S.O.I.).
Ook kunnen scholen contact opnemen met de:

Stichting Kinderen met Computers
Kuyperstraat 7
5694 CW Son en Breugel

Voor het speciaal en voortgezet speciaal onderwijs is er de Werkgroep Computers Speciaal Onderwijs. Het adres is:

Secretariaat W.C.S.O.
p/a Burg. van Hellenbergstraat 1
1217 LJ Hilversum
Tel. 035 - 15050

Aangesloten bij het W.C.S.O. werken een dertiental regionale groepen, COMBO's genaamd. Informatie hierover is te verkrijgen bij het W.C.S.O.

Overal ter wereld zijn mensen bezig met Logo. Om te horen wat al die mensen doen en om een poging te doen ze met elkaar in contact te brengen is er in de Verenigde Staten een Logo Penpal Club.
Het gaat om het uitwisselen van gedachten, ideeën en programma's onder Logo-gebruikers (vooral uit het onderwijs) over de gehele wereld. Het adres is:

Logo Penpal Club
University of Virginia
Ruffner Hall
405 Emmet Street
Charlottesville, VA 22905-2495
USA

EPILOOG

EPILOOG

Dit boek is geschreven vanuit de behoefte u, als lezer, iets meer over de prachtige programmeertaal Logo te vertellen. U heeft kunnen lezen dat Logo veel meer is dan het spelen met de Turtle. Aan de hand van zeer vele voorbeelden heeft u zich een beeld kunnen vormen van Logo.

Ik hoop dat u enthousiast bent geworden over het werken met Logo. De programma's die ik als illustratie gebruikte waren geen kant-en-klare programma's. Het is aan u om met deze programma's uw eigen programmeerervaringen op te doen. Elk van de gebruikte programma's kunt u naar hartelust veranderen en aanpassen aan uw eigen wensen. Ik heb slechts geprobeerd u de bouwstenen aan te reiken waarmee u zelf aan de slag kunt.

Daarom heb ik ook nooit de pretentie gehad 'volledig' te zijn. Vele zaken zijn onvermeld gebleven. Graag had ik bijvoorbeeld iets meer willen vertellen over de geweldige muziekmogelijkheden van Commodore Logo, of had ik wat dieper willen ingaan op het werken met lijsten in DR.Logo van de Atari 520/260. Enfin, een eerste aanzet is er in ieder geval.

Het schrijven dan dit boek was nooit mogelijk geweest zonder de hulp van enkele mensen. Ik wil ze hier even met name noemen. Bedankt Willem Heyster, voor het nakijken en corrigeren van het manuscript en voor het mogen gebruiken van de illustraties uit je werkboekjes; André de Regt, voor het maken van de prachtige illustraties voor het boek; Toon van Dijk, voor je deskundige hulp en advies betreffende Logo-machinetaalopdrachten, Joep Steeman en Peter Koster voor jullie correctiewerk, en Tetty voor alle geduld!

Bedankt ook iedereen die mij heeft gestimuleerd, gemotiveerd en geadviseerd.

Auke Sikma

LITERATUUR

- Logo Software Test, in *Happy Computer*, sept. 1984.
- Logo maakt leren leuk, in *P.C.M.*, febr. 1984.
- Logo Centrum Nederland, in *Basisbulletin*, april 1984.
- Kan dat ding ook aan?, in *Schildpadnieuws*, 1/1983.
- Logo, een elektronische zandbak, in *Schildpadnieuws*, 1/1983.
- Waarom eigenlijk Logo?, in *Schildpadnieuws*, 2/1983.

Allan, B., *Introducing Logo*, Granada, 1984.
Allan, B., *Pocket guide to programming Logo*, Pitman, 1984.
Allen, J.R., *Thinking about (TLC) Logo*, Holt Saunders, 1983.
Atari Logo, *Reference Manual*, Logo Computer Systems Inc., 1983.
Atari Logo, *Introduction to programming*, Logo Computer Systems Inc., 1983.
Baaijens, Nico, *Het denkende ding*, Het Spectrum BV, Utrecht/Antwerpen, 1981.
Berger, Joop, De schone belofte van Logo, in *Kijk*, aug. 1983.
Bolt, Steven, Zo wordt leren weer leuk, in *Kijk*, aug. 1983.
Burnett, D., *Logo: an introduction*, CCP 1983.
Camstra, B., *Leren en onderwijzen met de computer*, Stenfert-Kroeze, Leiden, 1980.
Dale, E.J., *Logo in the classroom*, Prentice Hall, 1984.
Davidson, L., *Apple Logo, reference manual*, Logo Computer Systems Inc., 1982.
Feigenbaum, Edward A., *De vijfde generatie*, Elsevier, Amsterdam, 1984.
Gervich, Jason, Managing the Atari Logo Workspace, in *Atari Explorer*, febr. 1985.
Gervich, Jason, Demons, turtles and things that ..., in *Atari Explorer*, april 1985.
Goldenberg, V. e.a., *Logo, a language for learning*, Terrapin Inc., 1982/1983.
Goodman, Danny, *Kinderen en computers*, Holkema & Warendorf, Weesp, 1985.
Goodyear, P., *A guide to learning through programming*, Heineman Ed. Books, 1984.
Greenfield, Patricia Marks, *Beeldbuiskinderen*, Intro, Nijkerk, 1986.
Greth, Carlos Vidal, Learning to learn, in *Atari Connection*, herfst 1983.

Greth, Carlos Vidal, In defense of Basic, in *Atari Connection*, herfst 1983.
Heyster, Willem, Leren programmeren?, in *De Computer Krant*, jan. 1985.
Heyster, Willem, Logo, een creatieve computertaal, in *AVRO-bode/ Televizier*, maart 1985.
Jansen, W.P., *Onderwijs en computers*, Malmberg, Den Bosch, 1984.
Lamers, H. en J.A. Wegkamp, *Computers in de basisschool*, Academic Service, Schoonhoven, 1986.
Lith, Willemien van, *Van schoolbord naar toetsenbord*, Van Gorcum, Assen/Maastricht, 1985
Ly, Eric, Player missile graphics, in *Atari Explorer*, juli 1985.
Maddison, A., *Computers op school*, Het Spectrum BV, Utrecht/ Antwerpen, 1984.
McLean, J., *Understanding Logo*, Alfred, 1984.
McBain, Craig, Turtle Piano, in *Antic*.
Morrow, J.T. e.a., Logo vs. Basic, in *Atari Connection*, herfst 1983.
Munniksma, Bob, Logo op de Atari 520 ST, in *Atari Info*, 1/1985.
Munniksma, Bob, Logo XL, in *Atari Info*, 1/1985.
Papert, Seymour, *Computers en kinderen*, Bert Bakker, Amsterdam, 1984.
Pinxteren, Harry, Een andere kijk op de computer, in *Info Media*, 3/1984.
Pinxteren, Harry, Computertaal voor kinderen, in *CPS blad*, 4/1984.
Pinxteren, Harry en Joop Ringelberg, *Logo*, Het Spectrum BV, Utrecht/Antwerpen, 1984.
Ross, Peter, *Logo programming*, Addison-Wesley, 1983.
Sikma, Auke, Logo, praktijkervaringen, in *Atari Info*, 2/1985.
Sikma, Auke, Computers en geestelijk gehandicapten, in *Onze Zorg*, okt. 1985.
Sikma, Auke, diverse artikelen, in *Atari Magazine*, 1/2/3/4/1985, 1/1986.
Sparrowhawk, Anne, *Logo*, Pan Books, London, 1984.
Thornburg, David D., The demons of Atari Logo, in *Compute*, jan. 1984.

INDEX

Enige opmerkingen.

- In het boek zult u niet alle Logo-primitieven beschreven vinden. In de Woordenlijst (pp. 202 e.v.) echter vindt u een opsomming van de meeste Logo-opdrachten en functies.

- De index heeft, per letter, de volgende indeling:
  - in HOOFDLETTERS de besproken Logo-primitieven en de voorbeeldprocedures; procedures zijn aangegeven met een *;
  - begrippen en trefwoorden die in de tekst voorkomen.

ACADEMIC SERVICE INFORMATICA UITGAVEN

AUTOMATISERING EN COMPUTERS

Computers en onze informatiemaatschappij - M.A. Arbib
Computers in de negentiger jaren - G.L. Simons
De informatiemaatschappij - Jan Everink
Op weg naar een risicoloze maatschappij? - Jan Holvast
Basiskennis informatieverwerking - Jan Everink
AIV, Automatisering van de informatieverzorging - Th.J.G. Derksen & H.W. Crins
BIV, Basis van de geautomatiseerde informatieverzorging - Th.J.G. Derksen & H.W. Crins
Organisatie, informatie en computers - David M. Kroenke
Computers in de basisschool - H. Lamers & J.A. Wegkamp
Effectieve toepassingen van computers - M. Peltu

MICROCOMPUTERS

Microcomputers thuis en op school - K.P. Goldberg & D. Sherwood
Bouw zelf een expertsysteem in BASIC - Chris Naylor
Programmeercursus MicrosoftBASIC - Nok van Veen
Werken met bestanden in BASIC - LeRoy Finkel & Jerald R. Brown
Programmeercursus BASIC op de Commodore 64 - Nok van Veen
Werken met bestanden op de Commodore 64 - G. Fisher, L. Finkel & J.R. Brown
Het Electron en BBC Micro boek - Jim McGregor & Alan Watt
Werken met bestanden op de Apple - LeRoy Finkel & Jerald R. Brown
Programmeercursus ApplesoftBASIC - Nok van Veen & Ad van Delft
Programmeercursus MSX BASIC - Nok van Veen
Werken met bestanden in MSX BASIC - LeRoy Finkel & Jerald R. Brown
40 grafische programma's - voor de Commodore 64; voor de Electron en BBC; voor de ZX Spectrum; voor de Apple II, IIe en IIc; in MSX BASIC - Marcel Sutter

MICROPROCESSORS EN ASSEMBLEERTALEN

Procescomputers, basisbegrippen - J.E. Rooda & W.C. Boot
Cursus Z-80 assembleertaal - Roger Hutty
6502 assembleertaal en machinecode voor beginners - A.P. Stephenson
EXAT-handboek - Micro-Teach

BESTURINGSSYSTEMEN

Inleiding besturingssystemen - A.M. Lister
Theorie en praktijk van besturingssystemen - J.L. Peterson & A. Silberschatz
Systeemprogrammatuur en softwareontwikkeling voor microcomputers - E. Verhulst
Bedrijfssystemen - EIT-serie, deel 4
CP/M, het operating system voor microcomputers - J.N. Fernandez & R. Ashley
CP/M 86, een besturingssysteem voor 16 bit microcomputers - J.N. Fernandez & R. Ashley
CP/M voor gevorderden - A. Clarke e.a.
PC DOS, het besturingssysteem van de IBM PC - R. Ashley & J.N. Fernandez
MS DOS, het besturingssysteem voor 16 bit microcomputers - R. Ashley & J.N. Fernandez
UNIX, het standaard operating system - G.J.M. Austen & H.J. Thomassen
De UNIX programmeeromgeving - B.W. Kernighan & R. Pike

PERSONAL COMPUTERS EN PROGRAMMAPAKKETTEN

De IBM PC en zijn toepassingen - Laurence Press
Werken met bestanden in IBM- en GW-BASIC - J.R. Brown & LeRoy Finkel
40 grafische programma's in IBM- en GW-BASIC - Marcel Sutter
Werken met VisiCalc - C. Klitzner & M.J. Plociak Jr.
Multiplan, een hulpmiddel bij de bedrijfsvoering - D.F. Cobb e.a.
Werken met Lotus 1-2-3 - G.T. LeBlond & D.F. Cobb
Lotus 1-2-3: Tips, Trucs en Tegenvallers - D. Andersen & D.F. Cobb
Lotus 1-2-3: Financiële macro's - Thomas W. Carlton
Symphony deel I en II - D.P. Ewing & G.T. LeBlond
dBASE III handboek - George Tsu-der Chou
WordStar stap voor stap - Ruth Ashley & Judi N. Fernandez

PROGRAMMEREN

Een methode van programmeren - Edsger W. Dijkstra & W.H.J. Feijen
Programmeren, met ontwerpen van algoritmen (met Pascal) - J.J. van Amstel
Voortgezet programmeren, het ontwerpen van datastructuren en algoritmen - J.J. van Amstel & J.A.A.M. Poirters

Problemen oplossen met de computer - R.G. Dromey
Inleiding tot het programmeren, deel 1 en 2 - J.J. van Amstel e.a.
Het Groot Pascal Spreukenboek - Henry F. Ledgard e.a.
Software engineering, het bouwen van grote programma's - I. Sommerville
JSP Jackson struktureel programmeren - Henk Jansen
JSP Uitwerkingenboek, JSP Procedureboek - Henk Jansen

PROGRAMMEERTALEN

Aspecten van programmeertalen - J.J. van Amstel & J.A.A.M. Poirters
Programmeertalen, een inleiding - J.J. van Amstel e.a.
Colloquium programmeertalen - red. J.A.A.M. Poirters & G.J. Schoenmaker
BASIC - EIT-serie, deel 3
Cursus BASIC, een practicum handleiding voor BASIC op de PRIME - R. Bloothoofd e.a.
Een programmeercursus in BASIC - Nok van Veen & René Wissing
Cursus Pascal - A. van der Sluis & C.A.C. Görts
Cursus eenvoudig Pascal - A. van der Sluis & C.A.C. Görts
Inleiding programmeren in Pascal - C. van de Wijgaart
Modula-2 - E. Verhulst
Systeemontwikkeling met Ada - Grady Booch
Cursus COBOL - A. Parkin
Cursus FORTRAN 77 - J.N.P. Hume & R.C. Holt
De programmeertaal C - L. Ammeraal
Flitsend Forth - Alan Winfield
Logisch LOGO - Auke Sikma
Programmeren in LISP - L.L. Steels
Micro-PROLOG, programmeren in logica - K.L. Clarke & F.G.McGabe
Inleiding PROLOG - W. Burnham & A. Hall

BESTANDSORGANISATIE, DATABASE EN GEGEVENSANALYSE

Bestandsorganisatie - R.J. Lunbeck & F. Remmen
Database, een inleiding - C.J. Date
Databases, grondslagen voor de logische structuur - F. Remmen
SQL in de praktijk - H.B. Eilers e.a.
Het SQL Leerboek - Rick F. van der Lans
Gegevensanalyse - R.P. Langerhorst

INFORMATIE-ANALYSE EN SYSTEEMONTWERP

Inleiding systeemanalyse en systeemontwerp - W.S. Davis
Systeemontwikkeling zonder zorgen - Paul T. Ward
Systeemontwikkeling volgens SDM - H.B. Eilers
Samenvatting SDM - Pandata
Systeemontwikkeling volgens JSD - Michael Jackson
Informatie-analyse volgens NIAM - J.J.V.R. Wintraecken
Information engineering - J. Blank
Het ontwerpen van interactieve toepassingen en computernetwerken - J.A. Scheltens
EDP Audit - C. de Backer
Management informatiesystemen - G.B. Davis & M.H. Olson
Prototyping, een instrument voor systeemontwerpers - red. T. Hoenderkamp & H.G. Sol
Het ontwikkelen van informatiesystemen met prototyping - R. Vonk
Simulatie, een moderne methode van onderzoek - S.K.T. Boersma & T. Hoenderkamp

EXPERTSYSTEMEN EN KUNSTMATIGE INTELLIGENTIE

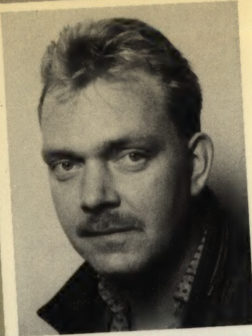
Kunstmatige intelligentie - Patrick H. Winston
Expertsystemen - Henk de Swaan Arons & Peter van Lith
Ontwikkelingen in expertsystemen - red. A. Nijholt & L.L. Steels

THEORETISCHE INFORMATICA EN SYSTEEMPROGRAMMATUUR

Systeemprogrammatuur - H. Alblas
Vertalerbouw - H. Alblas e.a.

OPERATIONELE RESEARCH

Operationele research - Y.M.I. Dirickx e.a.
Lineaire programmering als hulpmiddel bij de besluitvorming - S.W. Douma



Over de auteur

Auke Sikma is werkzaam als leerkracht op een school voor Speciaal en Voortgezet Speciaal Onderwijs. Daar werkt hij al geruime tijd met Logo als 'leeromgeving'. Daarnaast verzorgt hij demonstraties, publikaties en gastlessen over de programmeertaal Logo.

Over het boek

Logisch Logo is in eerste instantie bedoeld als inleiding in het gebruik van de krachtige programmeertaal Logo. Aan de hand van zeer veel voorbeelden laat de auteur de taal Logo van veel kanten zien: Logo als proceduretaal, als recursieve en interactieve taal. Bij het doornemen van de voorbeeldprocedures zult u, spelenderwijs, de logica van Logo ontdekken. Programmeren in Logo is dan ook vooral analytisch, creatief en probleemoplossend denken.
Bovendien is de drempel om met Logo te werken zeer laag. Zo laag zelfs, dat heel jonge kinderen al met Logo overweg kunnen. De auteur, zelf werkzaam in het onderwijs, geeft daarom ook veel aanwijzingen voor het gebruik van Logo met kinderen.
Is Logo dan een kindertaal? Volstrekt niet! Met Logo (dat zeer nauw verwant is aan LISP en PROLOG) zijn zeer geavanceerde procedures te schrijven.
De voorbeeldprocedures in dit boek zijn alle geschreven in LSCI Logo, o.a. voor Atari XL, maar zij zijn met behulp van de vergelijkingstabel achterin het boek voor iedere Logoversie geschikt te maken.

ISBN 90 6233 203 X